Herausgeber

H. SCHULTE

F. HOFFMANN

R. MIKUT



Berlin | 1. – 2. Dezember 2022

PROCEEDINGS **32. WORKSHOP** COMPUTATIONAL INTELLIGENCE



H. Schulte, F. Hoffmann, R. Mikut (Hrsg.)

Proceedings. 32. Workshop Computational Intelligence

Berlin, 1. – 2. Dezember 2022

PROCEEDINGS **32. WORKSHOP** COMPUTATIONAL INTELLIGENCE

Berlin, 1. – 2. Dezember 2022

Herausgegeben von
H. Schulte
F. Hoffmann
R. Mikut



Impressum



Karlsruher Institut für Technologie (KIT)
KIT Scientific Publishing
Straße am Forum 2
D-76131 Karlsruhe

KIT Scientific Publishing is a registered trademark
of Karlsruhe Institute of Technology.
Reprint using the book cover is not allowed.

www.ksp.kit.edu





Print on Demand 2022 – Gedruckt auf FSC-zertifiziertem Papier

ISBN 978-3-7315-1239-4
DOI 10.5445/KSP/1000151141

Inhaltsverzeichnis

Machine learning for estimating parameters of a convective-scale model: A comparison of neural networks and random forests

Stefanie Legler[1], Tijana Janjic[1], Mohammad Hossein Shaker[2], Eyke Hüllermeier[2]

[1]Mathematical Institute for Machine Learning and Data Science
KU Eichstätt-Ingolstadt
E-Mail: stefanie.legler@ku.de, tijana.janjic@ku.de

[2]Institute of Informatics
University of Munich (LMU)
Munich Center for Machine Learning (MCML)
E-Mail: mhshaker@mail.upb.de, eyke@lmu.de

1 Introduction

Errors and inaccuracies in the representation of clouds in convection-permitting numerical weather prediction models can be caused by various sources, including the forcing and boundary conditions, the representation of orography, and the accuracy of the numerical schemes determining the evolution of humidity and temperature. Moreover, the parametrization of microphysics and the parametrization of processes in the surface and boundary layers do have a significant influence. These schemes typically contain several tunable parameters that are either non-physical or only crudely known, leading to model errors and imprecision. Furthermore, not accounting for uncertainties in these parameters might lead to overconfidence in the model during forecasting and data assimilation (DA).

Traditionally, the numerical values of model parameters are chosen by manual model tuning. More objectively, they can be estimated from observations

by the so-called augmented state approach during the data assimilation [7]. Alternatively, the problem of estimating model parameters has recently been tackled by means of a hybrid approach combining DA with machine learning, more specifically a Bayesian neural network (BNN) [6]. As a proof of concept, this approach has been applied to a one-dimensional modified shallow-water (MSW) model [8].

Even though the BNN is able to accurately estimate the model parameters and their uncertainties, its high computational cost poses an obstacle to its use in operational settings where the grid sizes of the atmospheric fields are much larger than in the simple MSW model. Because random forests (RF) [2] are typically computationally cheaper while still being able to adequately represent uncertainties, we are interested in comparing RFs and BNNs. To this end, we follow [6] and again consider the problem of estimating the three model parameters of the MSW model as a function of the atmospheric state.

2 Model and methods

2.1 The MSW model

The MSW model is used to generate the true atmospheric state as well as the forecasts. This simple toy model realizes a mapping $x(t+dt) = MSW_\theta(x(t))$ to simulate the development of wind (u), clouds (h) and rain (r), and can be used to study new DA algorithms. A one dimensional grid with 250 grid points is used, yielding a state vector of the form:

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{u}(t) \\ \mathbf{h}(t) \\ \mathbf{r}(t) \end{bmatrix} \in \mathbb{R}^{750}. \tag{1}$$

In a realistic setting we do not have access to the true atmospheric state but only to observations which are sparse and noisy and only available at distinct times. We simulate this by adding noise to the true atmospheric state and only observe every 60 model time steps. Furthermore, we observe all variables of the MSW model only at those grid points where $r > 0.005$ to simulate radar

data. The MSW model parameters to be estimated are the rain removal rate α, the constant value for the geopotential ϕ_c, and the threshold for the fluid height h_r. All model parameters are constant in space and yield a three-dimensional parameter vector of the form

$$\theta(t_k) = \begin{bmatrix} \alpha(t_k) \\ \phi_c(t_k) \\ h_r(t_k) \end{bmatrix} \in \mathbb{R}^3. \tag{2}$$

that will be estimated in discrete times t_k. The parameters for training and testing are taken from uniform distributions with the same bounds as in [7, 6] and rescaled to the unit interval [0,1] before training.

2.2 Machine learning

A BNN and a RF regressor are used to estimate the three model parameters of the MSW model as a function of a snapshot in time of the atmospheric state (1). This results in an input size of 750 and an output size of 3. For the BNN, stochastic components are introduced over the weights [5] of a fully connected neural network with three hidden layers (see [6] for details of the architecture). The priors for the weights are normal distributions with mean 0 and standard deviation 1. These were optimized via ELBO-based stochastic variational inference [4] using Pyro [1]. The RF consists of 100 trees with the minimum sample size for a split set to five.

2.3 Data assimilation

In reality it is not possible to produce accurate weather forecasts by using only a dynamical model, but it is necessary to update the forecast at certain time intervals using current observations. This process is also called the DA cycle, and the updated forecast is called the analysis. Panels II. + III. in Fig. 1 outline the usual steps of such a DA cycle: at time t, a weather forecast for time $t+dt$ is generated using the MSW model which starts from the current analysis ensemble: $x^{fc}(t+dt) = MSW_\theta(x^{an}(t))$. Once the time $t+dt$ is reached, and

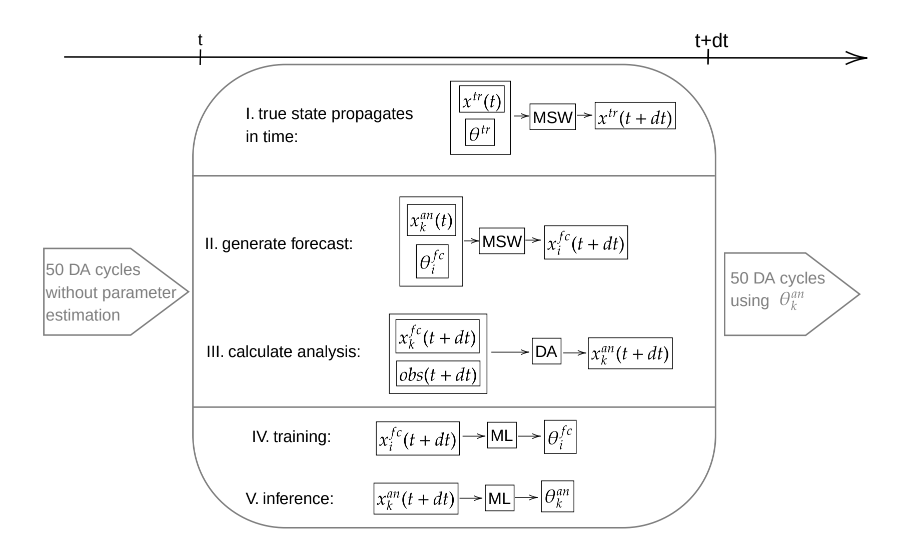


Figure 1: Sketch of the combined DA and ML algorithm to estimate the state and parameters.

thus observations of the true atmosphere are available, $x^{fc}(t+dt)$ is updated with the current observations obs_{t+dt} using a DA algorithm. We utilize a stochastic Ensemble Kalman Filter in this work [3]. For this algorithm a time-varying sample covariance is calculated and used for each forecast ensemble member, resulting in a new analysis ensemble $\{x_k^{an}(t+dt)\}$. This analysis ensemble can then be used to generate the next weather forecast for the time $t+2dt$. Note, if one would simply use $x^{fc}(t+dt)$ to generate the next forecast for $t+2dt$, instead of the analysis, the state error would grow over time and make the forecast inaccurate.

3 Experiments and results

For the experimental set-up outlined in Fig. 1, the true atmospheric state $x^{tr}(t)$ starts from a random initial state and is propagated in time by the MSW model using a sample of model parameters from the test set θ^{tr} which are kept constant in time. The objective is to estimate $x^{tr}(t)$ using DA methods and θ^{tr}

using ML methods. To simulate a realistic scenario in our toy model set-up, first 50 DA cycles with n_{ens} forecast and analysis ensemble members were generated without estimating parameters. For the first 50 DA cycles, the forecast model parameters are simply taken from the uniform distributions specified in the previous section. Then, a modified DA cycle takes place which incorporates the ML based parameter estimation (Figure 1: II.-V.). n_{train} parameters are sampled from the uniform distributions, resulting in the set $\{\theta_i^{fc}\}_{i=1}^{n_{train}}$. Each of these parameters represents one training label and is used to generate n_{train} forecast ensemble members (Figure 1: II.). Each forecast corresponds to one training input. Both ML methods are trained on the n_{train} input/label pairs (Figure 1: IV.) before they are used to estimate θ^{tr}. Because the ML algorithms are trained on the full state vector we would ideally use $x^{tr}(t)$ to infer its parameters. Since we do not have access to this state and the observations are sparse with a size smaller than the input size of the ML models we are using the current analysis ensemble (Figure 1: III.). Each analysis ensemble member is used to generate a set of 100 parameter estimates (Figure 1: V.) resulting in a set of $100 \times n_{ens}$ parameters. From this distribution n_{ens} parameters are sampled at each DA cycle for the next 50 cycles, resulting in a different subset $\{\theta_k^{an}\}_{k=1}^{n_{ens}}$ each time. The same experiment is run without the ML modified DA cycle to asses if the parameter estimation is able to reduce the error between the analysis ensemble and the true atmospheric state. Note, in [6] steps IV. + V. were repeated at every DA cycle. Since this is computationally quite expensive with only a small improvement over time we omitted those steps for this present work.

Figure 2 displays the means and standard deviations (std) of the parameter estimates for both ML methods for a training size of $n_{train} = 10000$ and an analysis ensemble size of $n_{ens} = 400$. While the averaged root-mean-square errors (RMSE) of the BNN is slightly smaller than that of the RF, the former tends to be a bit overconfident in its estimates, producing relatively low standard deviations even in cases where the parameter estimates are not very accurate. In this regard, the RF seems to quantify its uncertainty more adequately. Interestingly, there is a visible estimation bias toward intermediate parameter values, which can be observed for all three parameters and for both methods: low parameter values are systematically overestimated and high values are

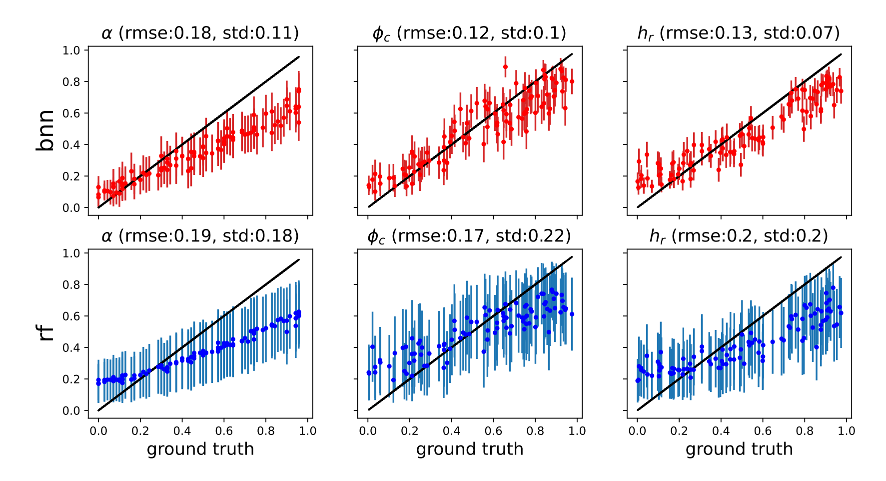


Figure 2: Scatter plot of means and standard deviations of parameter estimates from Bayesian neural network (first row) and random forest (second row) with 400 analysis ensemble members against ground truth for 100 samples of the test set.

underestimated. For the time being, the reason for this bias is not completely clear. We conjecture that it might be caused by the DA procedure we are using, which may not be optimal for convective-scale weather models and produce a discrepancy between forecasts/analysis and the true atmospheric states.

In Fig. 3, the RMSE at the end of the experiment is plotted against the number of ensemble members for the variables of the MSW model and for the parameters. In addition to the methods described above, two other experiments are shown, a first one where true parameter values were used and the state was estimated using DA, and a second one that uses random and false parameter values. For the estimation of parameters, the BNN is more accurate for almost all ensemble sizes. However, standard deviations when using BNN is much lower than RMSE, which is not the case for RF that shows similar values. For atmospheric state, the spread of the ensemble shows that all methods underestimate the RMSE for variables u and h. The situation is different for rain, where estimating parameters increases the uncertainty of the rain field to values higher than RMSE. This is the case for both, the estimation with BNN and RF.

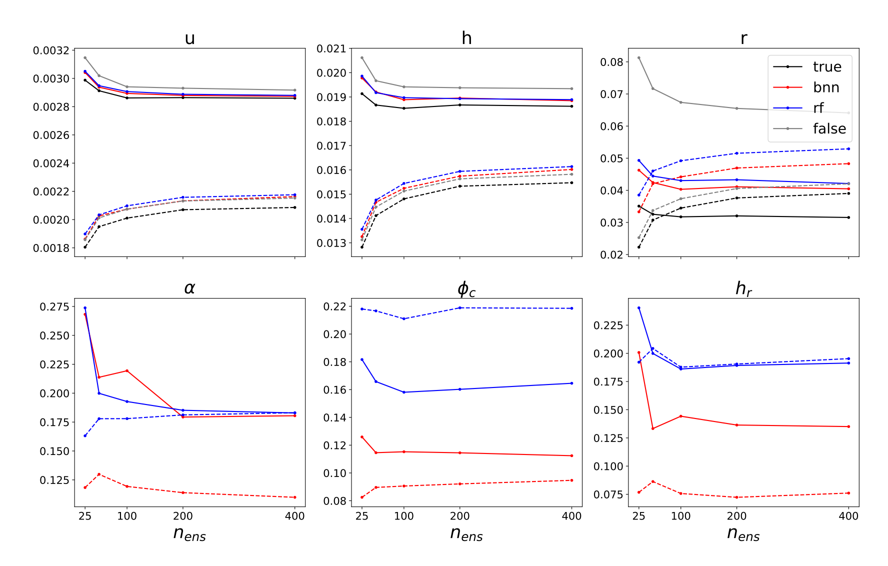


Figure 3: RMSEs (solid) and standard deviations (dashed) of analysis of atmospheric variables (first row) and parameter estimates (second row) against number of analysis ensemble members averaged over last 50 DA cycles and over 100 simulations with different ground truth parameters with $n_{train} = 10000$.

Finally, for the fixed ensemble size of 100, we show the sensitivity of the results to the number of training samples in Fig. 4. As seen there, for 100 members BNN needs at least $n_{train} = 5000$ to outperform RF for two parameters, while not even 10000 samples are enough for the rain removal rate α.

4 Conclusion

In this work, we compared Bayesian neural networks [5] and random forests for the estimation of parameters of the one-dimensional MSW model as a function of the analysis of the atmospheric state. Through perfect model experiments we show that both approaches are in principle able to estimate model parameters and to quantify the related uncertainty. However, while BNN seems to produce more accurate results on the test problems, the uncertainty estimates of RF are closer to RMSE values. For both methods, we observed a systematic estimation bias in boundary regions where parameters are very low or very high.

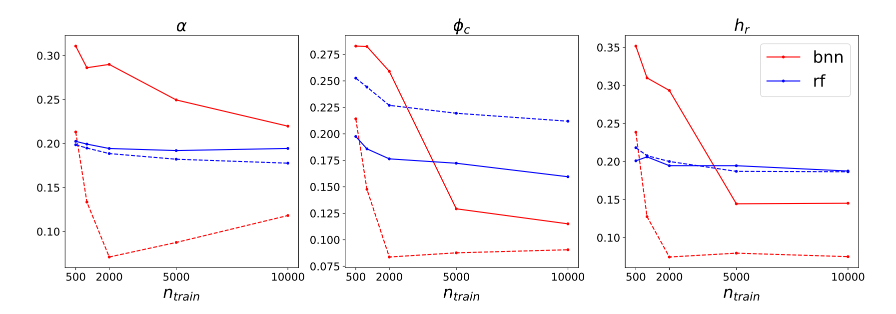


Figure 4: RMSEs (solid) and standard deviations (dashed) of parameter estimates against number of training samples averaged over 100 simulations with different ground truth parameters with $n_{ens} = 100$.

Moreover, the estimation of parameters combined with DA for the state decreases the initial state errors even when assimilating sparse and noisy observations.

References

[1] E. Bingham, J.P. Chen, M. Jankowiak, F. Obermeyer, N. Pradhan, T. Karaletsos, R. Singh, P.A. Szerlip, P. Horsfall, N.D. Goodman. "Pyro: Deep Universal Probabilistic Programming". In: *J. Mach. Learn. Res.,* 20:1-6, 2019.

[2] L. Breiman. "Random Forests". Machine Learning, 45(1):5–32, 2001.

[3] G. Evensen. "The Ensemble Kalman Filter: Theoretical Formulation and Practical Implementation". In: *Dynamics,* 53:343-367. 2003.

[4] M.D. Hoffman, D.M. Blei, C. Wang, J. Paisley. "Stochastic Variational Inference". In: *J. Mach. Learn. Res.,* 14:1303–1347. 2013.

[5] L.V. Jospin, H. Laga, F. Boussaid, W. Buntine, M. Bennamoun. "Hands-On Bayesian Neural Networks—A Tutorial for Deep Learning Users". In: *IEEE Computational Intelligence Magazine,* 17:29-48. 2022.

[6] S. Legler and T. Janjic. “Combining Data Assimilation and Machine Learning to Estimate Parameters of a Convective-Scale Model”. In: *Quarterly Journal of the Royal Meteorological Society,* 148(743):860–874. 2021.

[7] Y.M. Ruckstuhl and T. Janjic. “Parameter and State Estimation with Ensemble Kalman Filter-based Algorithms for Convective-scale Applications”. In: *Quarterly Journal of the Royal Meteorological Society,* 144(826–841). 2018.

[8] M. Würsch and G.C Craig. “A simple Dynamical Model of Cumulus Convection for Data Assimilation Research”. In: *Meteorologische Zeitschrift,* 23(5):483–490. 2014.

EasyMLServe: Easy Deployment of REST Machine Learning Services

Oliver Neumann, Marcel Schilling, Markus Reischl, Ralf Mikut

Institute for Automation and Applied Informatics,
Karlsruhe Institute of Technology, Hermann-von-Helmholtz-Platz 1,
76344 Eggenstein-Leopoldshafen
E-Mail: oliver.neumann@kit.edu

Abstract

Various research domains use machine learning approaches because they can solve complex tasks by learning from data. Deploying machine learning models, however, is not trivial and developers have to implement complete solutions which are often installed locally and include Graphical User Interfaces (GUIs). Distributing software to various users on-site has several problems. Therefore, we propose a concept to deploy software in the cloud. There are several frameworks available based on Representational State Transfer (REST) which can be used to implement cloud-based machine learning services. However, machine learning services for scientific users have special requirements that state-of-the-art REST frameworks do not cover completely. We contribute an EasyMLServe software framework to deploy machine learning services in the cloud using REST interfaces and generic local or web-based GUIs. Furthermore, we apply our framework on two real-world applications, i. e., energy time-series forecasting and cell instance segmentation. The EasyMLServe framework and the use cases are available on GitHub.

1 Introduction

Machine Learning (ML) approaches are able to solve complex tasks in various domains by learning from data, for example, instance segmentation [1], translation [2], or text-to-image generation [3]. Nonetheless, users struggle to apply ML approaches to their data because adapting code and setting up software and hardware environments need expert knowledge [4]. Therefore, it is important to deploy ML models in a way such that non-expert users can apply these models to their data easily.

Typically, ML models are deployed by programming Graphical User Interfaces (GUIs) because users are familiar with it [4, 5, 6]. This deployment strategy, however, has some disadvantages. Each individual user needs powerful hardware because ML models are computationally expensive. The hardware device is often underutilized as users do not process data continuously. Additionally, the hardware does not scale with user requests. Experts have to perform the installation process on the user site because of software dependencies, for example, GPU drivers or ML frameworks. Code quality suffers because of mixing model and GUI code. Updating software is complicated since distributed code snippets along different users.

To solve these deployment problems, we contribute a software framework (EasyMLServe) to deploy ML approaches as cloud-based software services using Representational State Transfer (REST) [7]. Therefore, we introduce a concept of cloud-based software, the framework architecture, and apply the framework to two real-world applications. Additionally, the framework is available on GitHub (`https://github.com/KIT-IAI/EasyMLServe`), including two real-world applications.

In Section 2, we introduce existing frameworks for the cloud-based deployment of ML models. The requirements, concept, architecture, and implementation of the contributed framework is explained in Section 3. In Section 3.3, we evaluate the frameworks and apply EasyMLServe on two real-world use cases, i. e., energy time-series forecasting and instance segmentation of biological images. Afterwards, we discuss and conclude the results in Section 5.6 and Section 6.

2 Related Work

There are several REST frameworks available to deploy ML services. A lot of them focus on high-performance like TorchServe [8] or TFX Serving [9]. Other frameworks offer easy-to-use config-based deployment of REST ML frameworks like DEEP as a Service (DEEPaaS) [10].

TorchServe is a framework to deploy ML services in the cloud and it is part of the PyTorch ecosystem [11]. The framework is actively maintained and allows parallel requests as well as advanced features like model performance optimization. TorchServe offers a broad range of examples due to the large community. However, TorchServe is restricted to the PyTorch ecosystem and excludes other ML frameworks like TensorFlow [12] or Scikit-Learn [13].

TFX Serving is part of the TensorFlow Extended (TFX) framework for deploying productive ML pipelines and is also part of the TensorFlow ecosystem. TFX Serving is also actively maintained and supports parallel requests, advanced features, and offers a variety of examples. However, like TorchServe, it is not independent of the ML framework, has a complex interface and documentation, and no GUI support.

DEEPaaS is an independent framework to deploy ML services in the cloud. It is actively maintained and it offers a simple config-based interface description. DEEPaaS has support for multiple workers but it does not handle multiple GPU access. Therefore, DEEPaaS recommends running only one worker if there is one GPU. Additionally, DEEPaaS offers no examples and no GUI support.

There are more REST frameworks for ML services available but all of them are very similar to the presented frameworks and have the same problems: They are focused on performance and, thus, are not independent of the ML framework. They offer many additional features, e.g., model management, which makes them complex, and they support no generic GUIs to support fast prototyping. With EasyMLServe, we want to offer an independent and easy-to-use REST framework for ML services that additionally deploys generative GUIs as a starting point for non-experts users.

3 EasyMLServe

In this section, we introduce the EasyMLServe framework to deploy ML services using REST APIs and generic GUIs. First, we define the requirements of REST frameworks for scientific ML services. Second, we describe the basic concept of REST APIs and how the data is exchanged between the ML service, the server, and the GUI. Third, we describe the most important EasyMLServe classes. Fourth, we explain how developers can deploy their ML approaches with the EasyMLServe framework.

3.1 Requirements

Developers in charge of implementing ML services for scientific users from different domains have requirements for REST frameworks to deploy their ML services.

- The REST framework should be actively maintained to ensure version compatibility, continuous improvement, and bug fixing.
- As developers of ML approaches for researchers, it is necessary to use different ML frameworks because not every novel ML approach is available in all ML frameworks. Therefore, a REST framework independent of the ML framework is required.
- In the research domain, prototypes of ML approaches need to be developed fast in the dynamic ML research community. Fast development can be achieved by offering easy-to-use and accessible frameworks.
- Non-expert users need a fast and easy way to use the ML approach. GUIs help to fulfill that objective.
- Real-world ML examples are needed to give developers a good starting point for their approaches. This makes the framework easier to use and reduces development time.

Other REST frameworks for ML services, in general, have additional requirements. However, those requirements are not needed in our use cases. Hence, we classify them as optional requirements:

- Industrial ML services need REST frameworks that handle thousands or more users in parallel which is challenging when thinking of large high-performance computing clusters on which the service should run. Examples of such applications can be found at Amazon, Spotify, or Netflix.
- Some REST frameworks offer more advanced features like model management that always come with additional code and configuration effort.

3.2 Concept

Based on the requirements, we propose a cloud-based service-oriented software architecture. Each ML approach is a software program (service) running on a remote computer (cloud). Data between services and users are exchanged using REST.

REST is a design principle for distributed systems and is based on the HTTP method stack [7]. Therefore, it can be used for every network, e. g., the internet or local private networks, and on any device, from smart meters to high-performance clusters. Software services that are based on REST often exchange data using JSON files which are equivalent to a list of key-value pairs.

If we apply the REST principle to our use case, we have a server offering a ML service by providing a REST interface. All communications between the GUI and server are based on HTTP messages. The REST interface, however, has no GUI. Therefore, our concept considers a GUI to communicate with the server via the REST interface. This GUI can be in form of a web page or a program running on a local computer. An overview of the data exchange between users and services is shown in Figure 1.

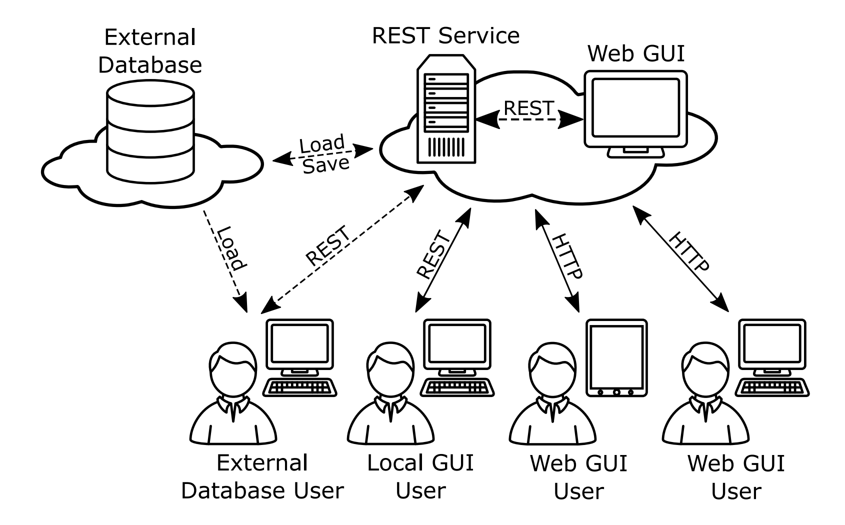


Figure 1: Data exchange between users and ML services. All data exchange between a user and a service is based on JSON objects. GUIs are deployed remotely as a web page or locally on a computer. Without GUIs, smart meters, or other low-end devices, can directly use the ML service using the REST interface. Additionally, data exchange between third-party actors is possible by using, for example, additional databases.

For the EasyMLServe framework, we decided that all requests and responses of the REST interface are encoded as JSON objects. It would be possible to exchange data directly as files but that increases the framework complexity. Therefore, all data from and to the ML services are encoded as JSON objects which makes it easier for developers and reduces framework complexity.

Depending on the GUI type, there are two ways of data exchange between users and the GUI. First, for web-based GUIs, we propose to communicate with the GUI via HTTP but not using a REST interface. Instead, the GUI is deployed using a web server that displays a web page to use the ML service. Second, for local GUIs, our concept suggests interacting directly with the GUI without using HTTP. In both cases, however, the GUI has to communicate with the ML service using the REST interface.

Data can also be exchanged with additional actors using any kind of communication protocol. A simple example would be to load and store data

from a database instead of uploading data via HTTP which can accelerate processing.

Regarding hardware constraints, the server needs to be deployed on powerful hardware depending on the ML task. For instance segmentation tasks, a Graphical Processing Unit (GPU) is recommended but for time-series forecasting using Linear Regression, a CPU is sufficient. The GUI, however, can be deployed on low-end hardware. Additionally, if a GUI is not needed, low-end devices can directly communicate with the REST interface. Smart meters, for example, can forecast energy time-series data or microscopes can process images by using high-performance computing clusters.

3.3 Architecture

The EasyMLServe framework consists of three major classes, i. e., *EasyMLService*, *EasyMLServer*, and *EasyMLUI*. The actual ML service is represented by the *EasyMLService* class. The *EasyMLServer* deploys the REST interface. *EasyMLUI* is the base class for all other UI classes. The relation between these three classes is shown in Figure 2.

EasyMLService is responsible for loading the model and processing JSON request of the REST interface. It returns JSON objects as a result of the REST interface request.

EasyMLServer provide the actual REST interface and passes all REST requests to the *EasyMLService*. Therefore, the *EasyMLServer* has to deploy a web server that handles HTTP messages which is done by the Uvicorn framework [15].

EasyMLUI is the base class for all available generic GUIs of the EasyMLServe framework. It handles the exchange of data between the user and the actual REST ML service. Therefore, *EasyMLUI* takes user input, prepares a REST request, and gets a REST response by passing the REST request to the REST ML service via HTTP. Currently, we support two frameworks for GUIs, i. e., PyQt [16] and Gradio [17]. Both GUIs are available using the child classes *QtEasyMLUI* and *GradioEasyMLUI*.

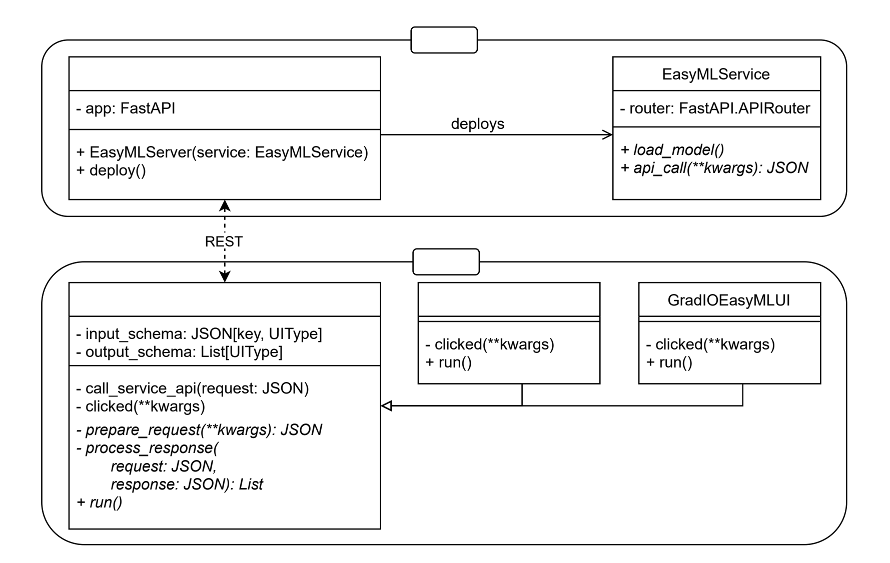


Figure 2: The basic architecture of EasyMLServe with the most important attributes and methods. The top block (Server) shows the relation between *EasyMLServer* and *EasyMLService*. The bottom part (GUI) shows the GUI classes and their relation. Both parts are separated such that an *EasyMLService* can be deployed by the server without using an *EasyMLUI* GUI.

3.4 Implementation

Developers that use EasyMLServe need to implement two classes: A service class with *EasyMLService* as parent class and a GUI class with *QtEasyMLUI* or *GradioEasyMLUI* as parent class. All relevant classes and their methods are shown in Figure 2.

EasyMLService consists of two methods developers need to implement, i. e., `load_model` and `api_call`. The `load_model` method is called after the *EasyMLService* is initialized and allows the user to load the model. The `api_call` method is called when a REST request is received with the actual JSON object. In code, developers have to implement a *EasyMLService* class like:

```
1 class MyMLService(EasyMLService):
2
3     def load_model(self):
4         # load and prepare model
5         pass
6
7     def process(self, request):
8         response ←{...} # prepare REST response
9         return response
```

EasyMLUI is the base class for all other GUI classes. Every GUI class has to implement two methods, i. e., prepare_request and `process_response`. The `prepare_request` method gets all user inputs defined in the input scheme and returns the REST request JSON encoded. The `process_response` method prepares and returns results that should be displayed to the user. In code, developers have to implement a *EasyMLUI* class like:

```
1 class MyMLServiceUI(EasyMLUI):
2
3     def prepare_request(self, some_ui_input):
4         request ←{...} # prepare REST request
5         return request
6
7     def process_response(self, request, response):
8         results ←... # prepare results (e.g. plots)
9         return results
```

To initialize *EasyMLUI* classes users have to define an input and output scheme. Input schemes describe the data users pass to the GUI, e. g., a CSV file. Output schemes determine what the GUI is presenting to the user, e. g., segmentation results or evaluation files. We need to define an input scheme as data can be passed in several ways, for example, text for translation tasks can be passed via text files or by typing text into text boxes. The output scheme is needed because displaying the response is often not sufficient, for example, when displaying segmentation results users may also be interested in the number of cells or mean cell size.

To define the input and output scheme, we implemented a set of *UITypes* which produce suitable GUI elements. These *UITypes* can be: *Text*, *TextLong*, *Number*, *Range*, *SingleChoice*, *MultipleChoice*, *File*, *ImageFile*, *CSVFile*, *TimeSeriesCSVFile*, or *Plot*.

After implementing an *EasyMLService* and *EasyMLUI* the resulting ML service and GUI can be deployed by calling the `run` method. In code, starting the server and service looks like:

```
1 # server.py
2 service ← MyMLService()
3 server ← EasyMLServer(service)
4 server.run()
```

```
1 # ui.py
2 input_schema ← {'some_ui_input': UIType()}
3 output_schema ← [Plot()]
4 app ← MyMLServiceUI(name← 'Example␣Service',
5                    input_schema← input_schema,
6                    output_schema← output_schema)
7 app.run()
```

4 Results

In the following, we evaluate the presented REST frameworks TorchServe [8], TFX Serve [9], DEEPaaS [10], and EasyMLServe based on the previously defined requirements for REST frameworks in scientific domains. Afterwards, we apply the EasyMLServe framework on two real-world applications, i. e.,

Table 1: REST frameworks for ML services evaluated on the necessary and optional requirements. Regarding necessary requirements, REST frameworks need to be actively maintained, independent of the ML framework, easy accessible (not complex), supports generative GUIs, and real-world examples. Regarding optional requirements, REST frameworks need to handle parallel requests and advanced features (e.g. model management).

Requirements	**REST Frameworks for ML**			
	TorchServe	TFX Serving	DEEPaaS	EasyMLServe
Maintained	✓	✓	✓	✓
Independent	✗	✗	✓	✓
Accessible	✗	✗	✓	✓
GUI Support	✗	✗	✗	✓
Examples	✓	✓	✗	✓
Parallel	✓	✓	(✓)	(✓)
Advanced Features	✓	✓	✗	✗

energy time-series forecasting and cell instance segmentation, to demonstrate the benefits of the proposed framework.

4.1 Evaluation

EasyMLServe is a REST framework for ML services that is focused on deployment of ML approaches for the research community. Therefore, we define specific requirements which need to be solved.

Evaluating existing REST frameworks with these requirements, we see that REST frameworks, which are focused on performance, are restricted to one ML framework, e. g., TorchServe or TFX Serve. Other ML frameworks like DEEPaaS, however, are independent of the ML framework but offer no generative GUI support.

Our EasyMLServe framework fulfills all requirements and closes the gap of actively maintained, ML framework independent, easy-to-use, and generative GUI supported REST frameworks. A comparison of the REST frameworks based on all requirements, including the optional ones, is shown in Table 1.

Additionally, to demonstrate the capabilities of EasyMLServe, we implemented two real-world use cases. First, electrical load forecasting for

Germany which is representative for several time-series ML problems. Second, biological cell instance segmentation which is a common ML task for image processing.

4.2 Time-Series Forecasting

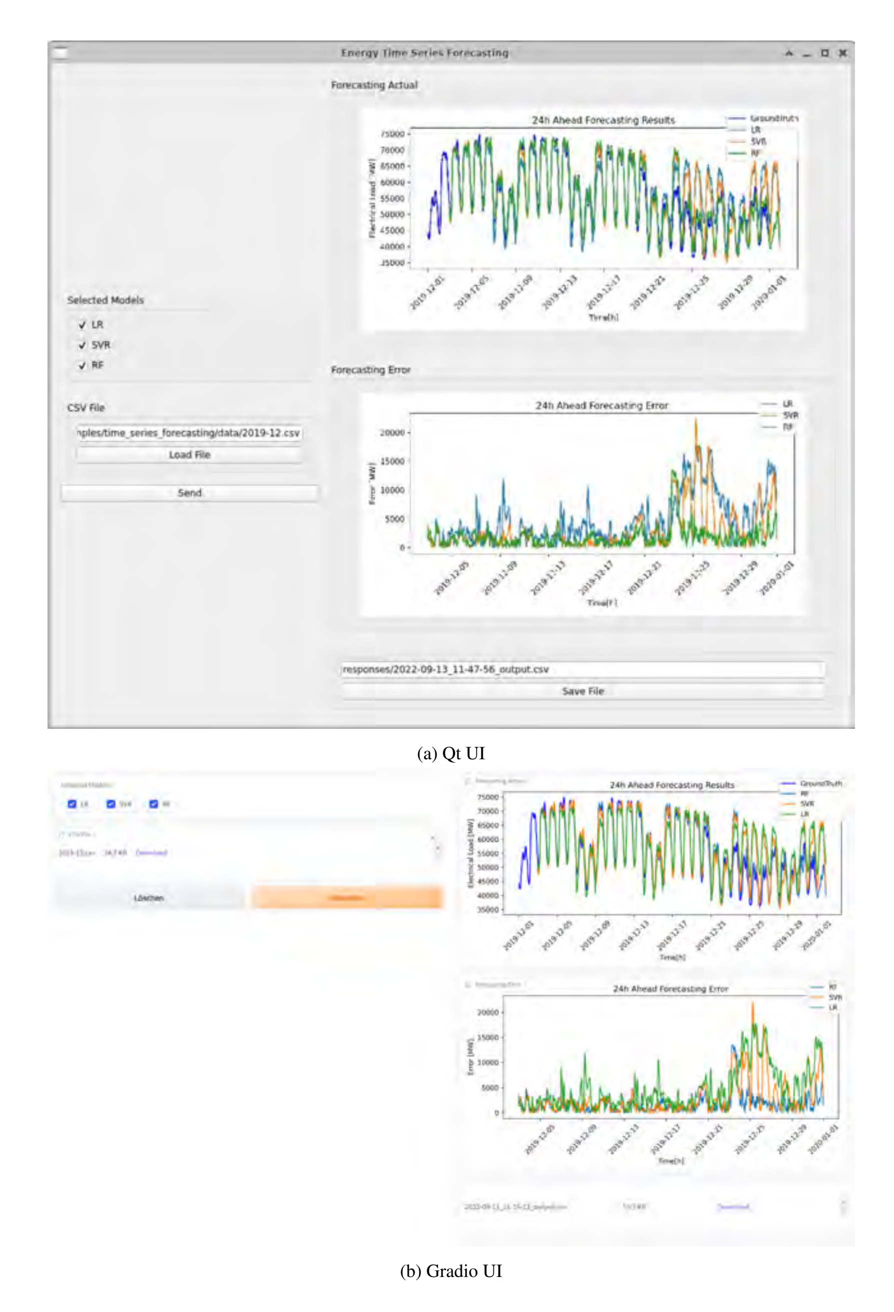
In the first use case, we forecast the electrical load for Germany one day ahead. Hourly electrical load data is used from the Open Power System Data (OPSD) dataset [14]. We used the years 2015 to 2018 for training. Regarding the models, we consider a Linear Regression, Support Vector Machine, and Random Forest model. All models are implemented using Scikit-Learn [13] with default parameter settings.

Our ML service expects a list of model names to use for the forecast, a list of time steps, and the corresponding energy values also as a list. After applying the selected models on the energy time-series, the ML service returns a forecast for each selected model in a list. Each forecast contains the corresponding model name, a list of time steps, and the corresponding energy values.

Regarding the GUI input scheme, we choose the MultipleChoice GUI element to define the model names. For the time steps and energy values, we use the CSVFile GUI element. This CSV file needs to be parsed to finally return the model names, time steps, and energy values as one JSON request. The parsing is done in the `prepare_request` method.

For the output scheme, we return two plots and a CSV file. For the two plots, we use the Plot GUI element to visualize the forecast and forecast error. Regarding the CSV file, a File GUI element which contains the actual one-day ahead forecasts is used.

Both supported GUIs can be deployed with EasyMLServe. Developers are able to switch easily between the GUIs by changing the parent class. The resulting GUIs are shown in Figure 3.



(a) Qt UI

(b) Gradio UI

Figure 3: Qt (a) and Gradio (b) based GUIs for the energy time-series forecasting use case. Qt is a locally deployed GUI. Gradio is a web-based deployed GUI.

4.3 Image Segmentation

The second real-world use case focuses on machine learning for images. We use microscopic images from the LIVECell dataset [18] and train a UNet [19] utilizing the KaIDA framework [20].

The ML service expects a Base64 encoded image containing the image encoding, data type, and shape. After segmenting the cells, our service returns an image of detected instances Base64 encoded. Note, the resulting ML service response has the same structure as the input request.

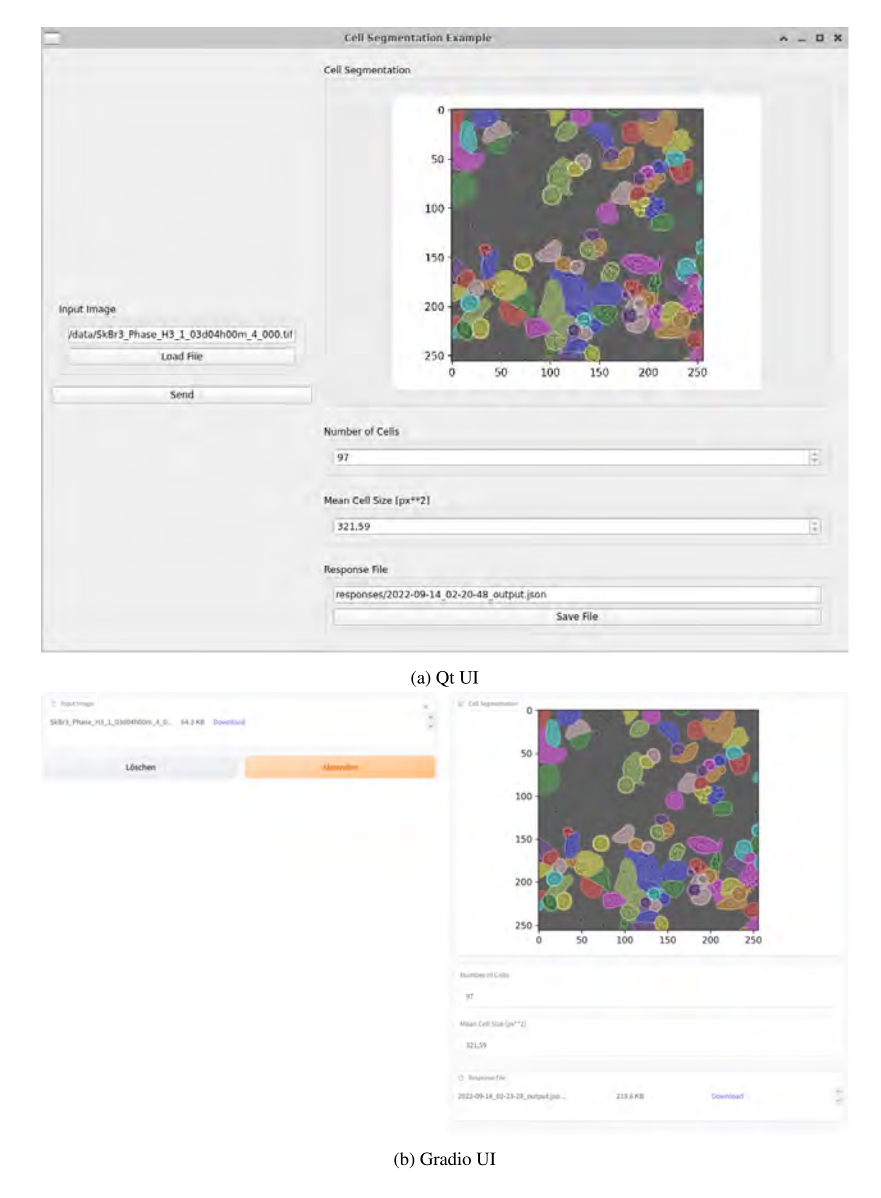
Regarding the GUI input scheme, we only use a File GUI element to select the image file. After loading the image, the image is encoded as Base64 and the REST request is created as JSON.

For the output scheme, we want to display the instances image, number of cells, mean cell size, and the ML service response as a JSON file. The instance overlay image is displayed by using the Plot GUI element of EasyMLServe. The number of cells and mean cell size are visualized using the Number GUI element. The response is displayed as a File GUI element where a user can download it.

The GUI generation framework of EasyMLServe supports Qt, as a local GUI, and Gradio, as a web-based GUI. Developers can switch between the two GUI frameworks by changing the parent class of the implemented *EasyMLUI* class. Both GUIs can be seen in Figure 4.

5 Discussion

EasyMLServe is a framework to easily deploy ML services using REST. To reduce complexity and make the framework as slim as possible we focus on the deployment part and leave out the training which is done by the developers in any environment. This has the disadvantage that users need help of experts in case they want to change the running ML service. Some frameworks offer training routines that allow users to retrain the models. However, we think also



(a) Qt UI

(b) Gradio UI

Figure 4: Qt (a) and Gradio (b) based GUIs for the cell segmentation use case. Qt is a locally deployed GUI. Gradio is a web-based deployed GUI.

retraining needs the supervision of experts. Especially in the research context where it is important that results are reliable.

The easy deployment of EasyMLServe includes generic GUIs. These GUIs support prototyping and fast deployment. The complexity of such GUIs, however, is limited. It can not cover all possible GUIs without losing its accessibility. Therefore, developers still have to develop GUIs on their own if the limited complexity of the generic GUIs is not enough.

In EasyMLServe, ML services exchange data using JSON objects which is a common way for REST services. It is also possible to directly upload files using multipart/form-data. This would avoid encoding and decoding files and thus reduces processing time and package size. We restricted the EasyMLServe REST interface to JSON objects because it made the framework and communication for the GUI easier. Furthermore, processing times of ML services are mostly restricted to the ML approach itself which is usually the most computational expensive part, e. g., instance segmentation using Deep Neural Networks running on a GPU.

For the implementation of EasyMLServe, we use Python as the programming language. Currently, the most common ML frameworks are written in Python. Therefore, all recent ML approaches are available in Python. However, ML approaches that are not written in Python can not easily be integrated into the presented framework.

Finally, EasyMLServe is a novel framework and we just started with a first version. There are bugs that need to be found and fixed as well as features which are currently missing. Nonetheless, bugs will appear and feature requirements will occur when developers apply this framework which belongs to a normal life cycle of software frameworks.

6 Conclusion

Scientific users have special requirements on the deployment of ML approaches. Deploying software solutions on-site has several disadvantages. Therefore, we propose a cloud-based solution that is based on REST and define

requirements of REST frameworks for scientific usage. These requirements are evaluated on the presented REST frameworks. Existing frameworks do not cover all necessary requirements completely and thus we contribute EasyMLServe, a REST framework for easy deployment of ML services in the cloud. Additionally, our presented framework offers generic GUIs for fast and easy prototyping.

EasyMLServe is a fast solution for ML developers to implement ML services in the cloud. It is actively maintained, independent of the ML framework, easy-to-use, supports generic local or web-based GUIs, and offers real-world applications as a starting point for developers.

To further improve EasyMLServe, we propose to deploy existing solutions with the EasyMLServe framework, for example, pyWATTS pipelines [21]. EasyMLServe is a novel framework which is under development. In future work, we aim to improve the EasyMLServe framework by fixing bugs and add additional features to enhance the user experience. This includes more GUI elements which need to be supported by the GUI generator as well as more complex GUIs.

Acknowledgments

This project is funded by the Helmholtz Association's Initiative and Networking Fund through Helmholtz AI, the Helmholtz Association under the Programs "Energy System Design"(ESD) and „Natural, Artificial and Cognitive Information Processing“ (NACIP), and the German Research Foundation (DFG) under Germany's Excellence Strategy – EXC number 2064/1 – Project number 390727645.

References

[1] C. Stringer, T. Wang, M. Michaelos, and M. Pachitariu, "Cellpose: a generalist algorithm for cellular segmentation," Nature Methods, vol. 18, pp. 100–106, 2021.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, et al., “Attention is All you Need,” in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[3] A. Ramesh, M. Pavlov, G. Goh, S. Gray, C. Voss, et al., “Zero-Shot Text-to-Image Generation,” in *International Conference on Machine Learning*, vol. 38 pp. 8821–8831, 2021.

[4] E. Gomez-de-Mariscal, C. Garcia-Lopez-de-Haro, W. Ouyang, L. Donati, E. Lundberg, et al., “DeepImageJ: A user-friendly environment to run deep learning models in ImageJ,” Nature Methods, vol. 18, pp. 1192–1195, 2021.

[5] S. Belda, L. Pipia, P. Morcillo-Pallarés, J. P. Rivera-Caicedo, E. Amin, et al., “DATimeS: A machine learning time series GUI toolbox for gap-filling and vegetation phenology trends detection,” Environmental Modelling & Software, vol. 127, p. 104666, 2020.

[6] C. Doty, S. Gallagher, W. Cui, W. Chen, S. Bhushan, et al., “Design of a graphical user interface for few-shot machine learning classification of electron microscopy data,” Computational Materials Science, vol. 203, p. 111121, Feb. 2022.

[7] R.T. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” *Doctoral dissertation*, University of California, Irvine, 2000.

[8] H. Bafna, et al., “TorchServe a flexible and easy to use tool for serving and scaling PyTorch models in production,” on *GitHub*, `https://github.com/pytorch/serve`, accessed: Sep. 2022.

[9] K. Gorovoy, et al., “TensorFlow Serving a flexible, high-performance serving system for machine learning models, designed for production environments,” on *GitHub*, `https://github.com/tensorflow/serving`, accessed: Sep. 2022.

[10] A. Lopez Garcia, “DEEPaaS API: a REST API for Machine Learning and Deep Learning models,” Journal of Open Source Software, pp. 1517, 2019.

[11] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[12] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," Software available at tensorflow.org, 2015.

[13] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, et al., "Scikit-learn: Machine Learning in Python," Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[14] F. Wiese, I. Schlecht, W.D. Bunke, C. Gerbaulet, L. Hirth, et al., "Open Power System Data – Frictionless data for electricity system modelling," Applied Energy, vol. 236, pp. 401–409, 2019.

[15] T. Christie, et al., "Uvicorn an ASGI web server implementation for Python," on *GitHub*, `https://github.com/encode/uvicorn`, accessed: Sep. 2022.

[16] Riverbank Computing Limited, The QT Company, "PyQt6 Reference Guide," on web, `https://www.riverbankcomputing.com/static/Docs/PyQt6/`, accessed: Sep. 2022.

[17] A. Abid, A. Abdalla, A. Abid, D. Khan, A. Alfozan, et al., "Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild," arXiv, 2019.

[18] C. Edlund, T. R. Jackson, N. Khalid, N. Bevan, T. Dale, et al., "LIVECell – A large-scale dataset for label-free live cell segmentation," Nature Methods, vol. 18, pp. 1038–1045, 2021.

[19] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," in *Medical Image Computing and Computer-Assisted Intervention* pp. 234–241, Springer International Publishing, 2015.

[20] M. P. Schilling, S. Schmelzer, L. Klinger, and M. Reischl, "KaIDA: a modular tool for assisting image annotation in deep learning," Journal of Integrative Bioinformatics, pp. 20220018, 2022.

[21] B. Heidrich, A. Bartschat, M. Turowski, O. Neumann, K. Phipps, et al., "pyWATTS: Python Workflow Automation Tool for Time Series," arXiv, 2021.

Multi-stage Inspection of Laser Welding Defects using Machine Learning

Patricia M. Dold[1,2], Fabian Bleier[1], Meiko Boley[1],
Ralf Mikut[2]

[1] Bosch Research
Robert Bosch GmbH
Robert-Bosch-Campus 1, 71272 Renningen
E-Mail: patricia.dold@de.bosch.com
[2]Institute for Automation and Applied Informatics
Karlsruhe Institute of Technology
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen

Abstract

As welding processes become faster and components consist of many more welds compared to previous applications, there is a need for fast but still precise quality inspection. The aim of this paper is to compare already existing approaches, namely single-sensor systems (SSS) and multi-sensor systems (MSS) with a proposed cascaded system (CS). The introduced CS is characterized by the fact that not all available data are analyzed, but only cleverly selected ones. The different approaches consisting of neural networks are compared in terms of their accuracy and computational effort. The data are recorded from scratch and include two common sensor systems for quality control, namely a photodiode (PD) and a high-speed camera (HSC). As a result, when the CS makes half of the final decisions based on a SSS with PD signals and the other half based on a SSS with HSC images, the estimated computational effort is reduced by almost 50% compared to the SSS with HSC images as input. At the same time, the accuracy decreases only by 0.25% to 95.96%. Additionally, based on the CS, a general cascaded system (GCS) for quality inspection is proposed.

1 Introduction

Laser welding is widely used in automotive, aerospace or shipbuilding industries and is considered a key technology in manufacturing [1, 2, 3]. Advantageous compared to other joining techniques is its ability for precise and fast welding. Unfortunately, laser welding, in which a workpiece is melted, vaporized and solidified, often is a challenging process [4] and, hence, welding defects occur. Especially in the electric drive train, some components consist of several hundred laser-welded elements, and a single welding defect can lead to the failure of the entire component. In order to detect defective elements in time, there is a desire in the industry for quality monitoring.

The most important sensor methods for examining weld quality include photodiodes (PD) [5, 6, 7] or spectrometers [8, 9] due to their simple structure and low cost. Other methods to provide information about spatter, keyhole, weld pool or plasma are ultraviolet sensors [10], X-rays [11, 12], microscopy, optical coherent tomography (OCT) [13] or high-speed cameras (HSC) [14, 15, 16, 17]. Accordingly, compared to typical quality monitoring in resistance welding [18] or ultrasonic welding [19], there are more potential input variables in laser welding.

The signals acquired by the different sensors are analyzed using signal or image processing algorithms. Thereby, data-driven process monitoring has been implemented by applying machine learning methods such as support vector machines (SVM) [20], decision trees [21], random forest algorithms [22] or Bayes classifiers [23]. Recently, deep learning has achieved great success in image recognition and classification [24] and thus has been applied to weld defect inspection, especially convolutional neural networks (CNN).

Some researchers use a single sensor type to study a specific mechanism of the welding process: In [6] the optical intensity captured by a PD when welding defects occur is analyzed. [17] does quality assessment of welds based on HSC or OCT data using deep neural networks like Inception-v3 [25]. Different CNNs like AlexNet, VGG-16 or MobileNetV3-Large [26] are used in [27] for automated optical inspection of laser welding.

However, information captured by one sensor is not sufficient for holistic quality assessment [28]. A combination of different sensor types, on the other hand, provides a more comprehensive description of the welding process [11]. In [29] different sensor types are used, including an ultraviolet-visible band visual sensor system, a spectrometer or a PD in order to detect three different weld defects during high power disk laser welding using neural networks. In [30] also several sensors in order to predict the welding quality are applied using machine learning methods.

A quality-monitoring system whose processing time is not longer than its cyclic time would be optimal. Yet, complex multi-sensor systems and processing algorithms can result in quality-monitoring systems that are not used in production due to long evaluation times. For this reason, the present paper proposes a cascaded system (CS) with the aim of fast but still precise quality inspection. The system follows a multi-stage structure: The first level of inspection has time series as input, with the advantage of a high clock rate as well as low memory requirements. This stage already safely classifies some welds; in areas of uncertainty, on the other hand, the next more complex stage with image data as input takes over in order to make a final decision. Quality control based on two-stages has already been applied to other use cases [23, 31, 32, 33]. In [23] a two-stage classifier for solder joint inspection has been proposed. After feature selection based on the algorithm of Bayes, each solder joint is classified by its qualification. If the solder joint fails in a qualification test, it is classified based on a SVM. Moreover, in [33] an inspection system for ball bonding is incorporated using CNNs or SVMs, where human judgment is only used when the detection uncertainty is below a threshold. The present paper deals with a cascaded system for quality control. Thereby, it links to already existing ideas of two-stage process monitoring in the literature. To show the added value of the system presented in this paper, it is compared to a single-sensor and multi-sensor systems. It lays the foundation for further cascaded decision-making systems in quality control.

2 Data Set

Since public data sets are usually not available for industrial research, especially for welding applications, the data used in this paper were acquired from scratch in the laboratory. Photodiode signals and associated high-speed camera images captured during the welding of 60 pairs of metal plates are considered. Subsequently, the recorded data were preprocessed so that they can be used by the various quality monitoring approaches considered in this paper.

2.1 Experimental Setup

In the considered laser welding process, two metal plates with a thickness of 75 μm were welded together with a power of 250 W in a commercial setup. The plates were placed on top of each other, clamped and welded in a rectangular geometry. Figure 1a illustrates the welding process of the two metal plates in cross-section. Figure 1b shows the rectangular welding path. The metal plate pair is viewed from above, which means that the plates are on top of each other in the image plane. Different anomalies such as spatters or gaps were manually inserted into the welding process. Of 60 welded plates, anomalies were inserted in 51, whereas 9 were welded under reference conditions. The dashed line indicates the position of anomalies. The initial position of the laser beam was $(x,y) = (0,0)$. It was then deflected to position $(x_{\text{on}}, y_{\text{on}})$ where the laser starts to weld the geometry and ends in position $(x_{\text{off}}, y_{\text{off}})$. Finally, the laser beam returned to its initial position and was ready for the next weld. In total, the welding process for a pair of metal plates took 338 ms.

Figure 2 shows the experimental setup. The laser optics were located above the metal plates at $(x,y,z) = (0,0,40\,\text{cm})$ and directed the laser beam to the rectangular path using two mirrors. A photodiode (PD) measured light with a wavelength of about 300–950 nm obtained during the welding process in the area of the weld pool. Accordingly, at each sampling time, the voltage of the PD amplifier was recorded. Furthermore, HSC images were taken during the process. The PD had a sampling rate of 250 kHz, whereas the HSC had one of 20 kHz. The advantage of the PD is the higher sampling rate as this allows

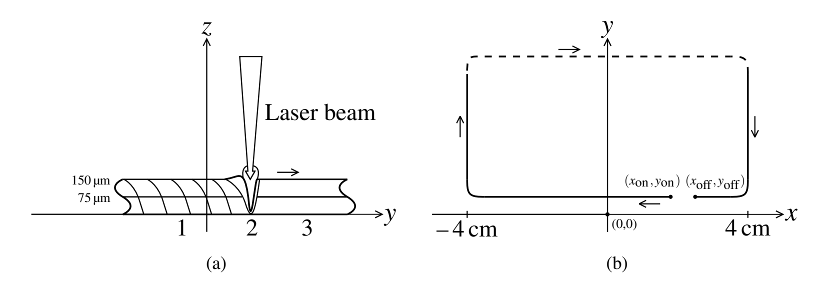


Figure 1: Laser welding process. (a) shows the two metal plates in cross-section during the welding process. There are three areas: 1 solidified melt, 2 weld pool and 3 unwelded metal plates. In (b) the rectangular welding path is indicated. Dashed lines indicate where anomalies were introduced in the laboratory experiment. Arrows indicate the welding direction.

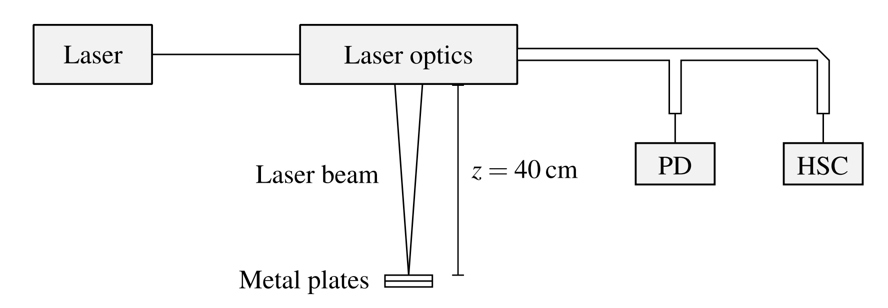


Figure 2: Experimental setup of the laser welding process of two metal plates. The two used sensors for data acquisition are a photodiode (PD) and a high-speed camera (HSC).

the detection of shorter anomalies compared to the HSC. Moreover, faster data processing is possible due to the smaller amount of raw data compared to the HSC. However, the HSC images provide information which is not available in the PD signals, e.g. information about the current geometry of the interaction zone. Figure 3 illustrates the sampling rates and the data of both sensors. The gray lines on the x-axis correspond to the sampling times of the PD and have a distance of 4 µs each. The black larger lines on the axis indicate the sampling times of the HSC, which are 50 µs apart. Every 12.5 samples of the PD are followed by a HSC gray scale image. In addition to the sampling rates, the PD and HSC data are visualized. In the range from 500–700 µs data of reference welds are shown. At $t = 250\,000$ µs an anomaly occurs, more precisely, the process spatters. On closer inspection, a slightly different interaction zone

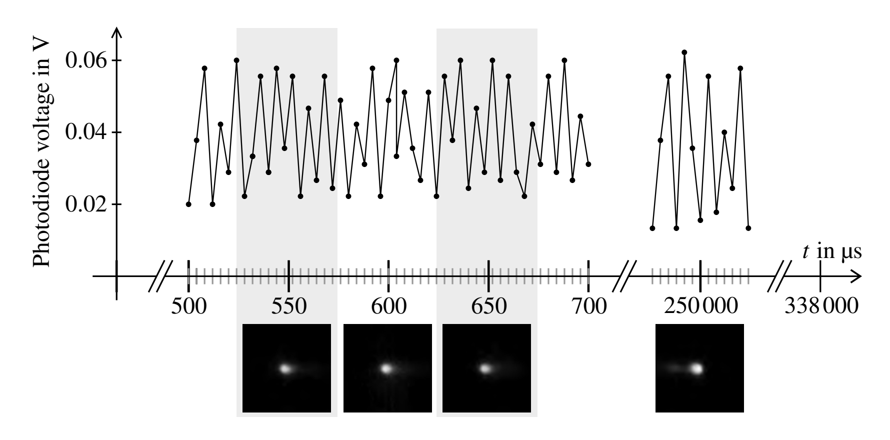


Figure 3: Illustration of the PD and HSC data and sampling rates.

geometry and a splash on the left side can be seen compared to the reference HSC images.

2.2 Preprocessing

For the quality-monitoring approaches considered in this paper, 13 samples of the PD were assigned to each image of the HSC. The assignment of the PD samples to the corresponding HSC image is visualized in Figure 3 by the gray background. Besides, the gray scale HSC images were cropped to a section of 100×100 pixels and scaled to a value range of [0,1].

A label was manually assigned to each image with the corresponding 13 PD samples, distinguishing between reference and anomaly. By section-wise labeling of the weld seams, a localization of reference and anomaly on the welding path is possible. This would not be possible when labeling a pair of plates as a whole. An anomaly refers to those locations, where an abnormality like a gap or spatters have been introduced. Reference is defined as those places where neither an abnormality is provoked nor is visible in the recorded PD signals or in the HSC images. There are places, where no intentional defects were introduced, but abnormalities are visible in the signal. One cause is sporadically occurring errors. Moreover, micrograph analyses show that

Table 1: Number of samples of the recorded data set. During training, data from 48 metal plates (80%) is used; during testing, data from 12 plates (20%) is used. The number of samples n_{DA} generated by data augmentation (DA) is also given. The values given indicate the average sample numbers over the 5 folds of the used cross-validation.

Label	n_{train}	n_{test}	n_{total}	n_{DA}
Reference	148 425	37 106	185 531	1 576 738
Anomaly	152 777	38 194	190 971	1 623 262
Total	301 202	75 300	376 502	3 200 000

anomalies may be present in the weld at that places. These positions are left out in the following because the correct label cannot be determined and would therefore confuse the process monitoring systems. Of the 60 metal plate units, the proportion of such identified locations is 7.19%.

Table 1 shows the number of samples of the data set used in this paper. During training, data from 48 metal plates were used and during testing, data from 12. For the evaluation of the models, a 5-fold cross-validation was applied. Thereby, in every fold the data of 12 metal plates were left out. Since different numbers of samples were left out on each metal plate, the number of samples varied in every fold. The average number of training samples over the 5 folds is given by n_{train}; the average number of test samples by n_{test} and both together as n_{total}. In order to teach the models invariances and robustness properties, a data augmentation on the PD signals as well as on the HSC images was used. The PD signals were reflected horizontally. The HSC images were rotated and reflected. This data augmentation is essential for the HSC images since, as shown in Figure 1b, anomalies were only inserted on one side of the welded geometry. If not applying this data augmentation, the classification algorithms could learn, that anomalies only take place in one direction, which is not the case in real processes.

3 Different Approaches for Quality Monitoring

Three different approaches, namely single-sensor systems (SSS), multi-sensor systems (MSS) and cascaded systems (CS) are considered. Before the approaches are introduced, the formal definitions of the analyzed data are defined:

For each sample, the time series of the PD signal corresponding to a HSC image is defined as

$$X_{\mathrm{PD},i} = \begin{pmatrix} x_{i,1} & x_{i,2} & \dots & x_{i,13} \end{pmatrix}, \tag{1}$$

where $i \in \{1, \dots, 376\,502\}$ indicates the considered sample number. The images of the HSC are defined as

$$X_{\mathrm{HSC},i} = \begin{pmatrix} x_{i,1,1} & x_{i,1,2} & \cdots & x_{i,1,100} \\ x_{i,2,1} & x_{i,2,2} & \cdots & x_{i,2,100} \\ \vdots & \vdots & \ddots & \vdots \\ x_{i,100,1} & x_{i,100,2} & \cdots & x_{i,100,100} \end{pmatrix}. \tag{2}$$

With the label $Y_i \in \{0,\ 1\}$, where 0 stands for anomaly and 1 for reference, the data set S consists of 376 502 triples according to

$$S = \{(X_{\mathrm{PD},1}, X_{\mathrm{HSC},1}, Y_1), \dots, (X_{\mathrm{PD},376\,502}, X_{\mathrm{HSC},376\,502}, Y_{376\,502})\}. \tag{3}$$

3.1 Single-Sensor and Multi-Sensor System

A single-sensor system (SSS) performs process monitoring based on data coming from one sensor. This can be represented by the function

$$f_{\mathrm{SSS}} : \mathbb{R}^{a \times b} \to \{0,1\} : X_i \mapsto Y_i, \quad a, b \in \mathbb{R}. \tag{4}$$

Figure 4 illustrates SSSs based on PD signals or HSC images, where $\hat{f}_{SSS}$ indicates an optimized model. It could be any classical machine learning algorithm

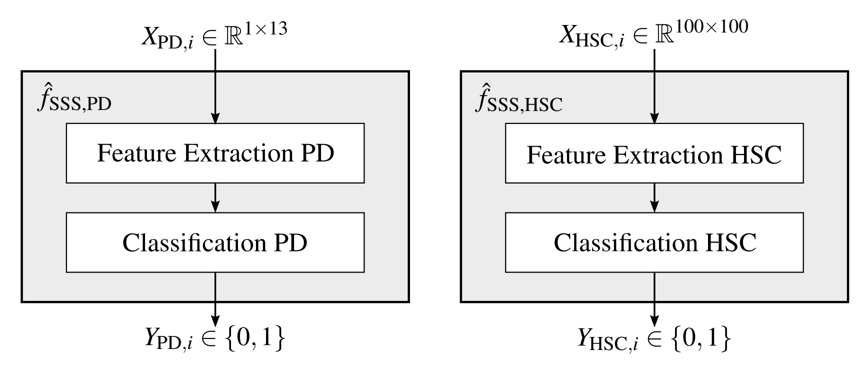


Figure 4: Two single-sensor systems (SSS). Thereby, $\hat{f}_{\text{SSS,PD}}$ and $\hat{f}_{\text{SSS,HSC}}$ are the prediction models based on PD samples $X_{\text{PD},i}$ or HSC samples $X_{\text{HSC},i}$ as input. $Y_{\text{PD},i}$ and $Y_{\text{HSC},i}$ are the predicted labels, which deviate from Y_i in case of misclassification.

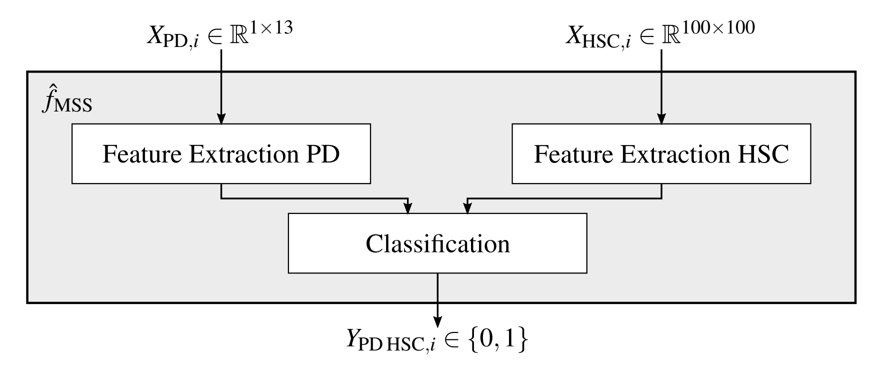


Figure 5: Multi-sensor system (MSS). The prediction model $\hat{f}_{\text{MSS}}$ has PD samples $X_{\text{PD},i}$ and HSC samples $X_{\text{HSC},i}$ as input and predicts $Y_{\text{PD HSC},i}$.

or a neural network with convolutional layers for feature extraction and fully-connected layers for decision-making. In the case of a neural network, the output is first a real number, which is then binarized for the final decision. Those predicted labels differ from Y_i in the case of misclassification.

A multi-sensor system (MSS) uses data from multiple sensors for process monitoring. A MSS consisting of PD signals and HSC images as inputs can be expressed as

$$f_{\text{MSS}} \colon \mathbb{R}^{1\times 13} \times \mathbb{R}^{100\times 100} \to \{0,1\} \colon X_{\text{PD},i} \times X_{\text{HSC},i} \mapsto Y_i. \tag{5}$$

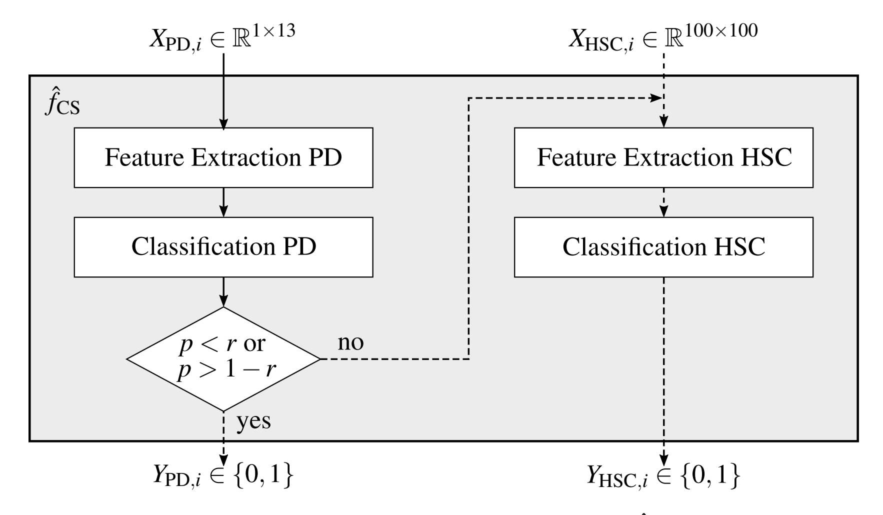


Figure 6: Cascaded system (CS). Depending on p, the prediction model $\hat{f}_{\text{CS}}$ decides either for $Y_{\text{PD},i}$ or $Y_{\text{HSC},i}$.

Figure 5 represents a MSS based on PD signals and HSC images. The advantage of a MSS compared to a SSS is a more holistic quality control. However, the computational effort of the MSS is greater than of the SSS, which is disadvantageous for industrial quality monitoring of fast laser processes.

3.2 Cascaded System

A cascaded system (CS) offers the possibility to use several sensors as a MSS does. In contrast to MSS, on the other hand, it is characterized by the fact that not all available data are analyzed to get the quality-weld condition; only cleverly selected data are evaluated. Figure 6 shows the two-stage CS used in this paper. Let $p \in [0, 1]$ be the output of a classifier which makes quality assessment based on PD signals. For $p < 0.5$ the classifier chooses anomaly; for $p \geq 0.5$ reference. The closer p is to 0 or 1, the more confident the classifier's decision is considered to be. Let $r \in (0, 0.5)$ be a fixed threshold. A decision of the classifier is evaluated as certain if $p < r$ or $p > 1 - r$. If the first condition is satisfied, the classifier is sure that it is an anomaly; if the second is satisfied, it is certain that it is a reference. If the classifier is sure, the result

is accepted; if not, a final quality decision is made in a next step based on the HSC data. Formally, this can be expressed by

$$f_{\mathrm{CS}}: \begin{cases} f_{\mathrm{SSS,PD}}, & \text{if} \quad p < r \text{ or } p > 1-r \\ f_{\mathrm{SSS,HSC}}, & \text{otherwise.} \end{cases} \tag{6}$$

4 Network Structures and Training Details

The four considered prediction models $\hat{f}_{\mathrm{SSS,PD}}$, $\hat{f}_{\mathrm{SSS,HSC}}$, $\hat{f}_{\mathrm{MSS}}$ and $\hat{f}_{\mathrm{CS}}$ were built by combining four blocks, namely Feature Extraction PD, Feature Extraction HSC, Classification PD and Classification HSC. The blocks consist of neural network architectures, which are described in detail later. Building the four models based on the same blocks provides the advantage of better performance comparison. $\hat{f}_{\mathrm{SSS,PD}}$ and $\hat{f}_{\mathrm{SSS,HSC}}$ use the combination of two blocks each according to Figure 4. $\hat{f}_{\mathrm{MSS}}$ was constructed according to Figure 5 using Classification HSC as classification block. $\hat{f}_{\mathrm{CS}}$ uses the combination of the four neural network blocks according to Figure 6.

The Feature Extraction PD block is based on [34], where different deep neural networks for time-series classification are proposed. It consists of four 1D convolution layers with 8, 16, 16 and 8 filters respectively and a kernel size of 3. Each convolution layer is followed by batch normalization and a ReLU as activation function. The Feature Extraction HSC block is based on MobileNet [35]. It consists of the MobileNet architecture until the last depthwise separable convolution layer. Pretrained weights based on ImageNet were used. The Classification PD block has three fully-connected layers with 16, 8 and 4 neurons followed by the single output gained with a sigmoid activation function for classification. Every fully-connected layer has a ReLU as activation function followed by a dropout layer with a rate of 0.5. The Classification HSC block consists of the same structure as Classification PD but with the fully-connected layers having 256, 256 and 128 neurons each.

During training of the models, the binary cross-entropy was used as loss function. An Adam optimizer with a learning rate $l_r = 5 \cdot 10^{-5}$ and the regularizers $\beta_1 = 0.9$ and $\beta_2 = 0.99$ were used. The models were trained with a batch

size of 32 with 100 steps per epoch for 1000 epochs. All models were trained with the same seed, meaning that every model is trained with the exact same augmented images or time series respectively in the exact same order for better performance comparison. Additionally, 5-fold cross-validation was used to get the final results. The models were implemented in Python using Keras and Tensorflow. The training processes ran on a NVIDIA A40 GPU card.

5 Results and Discussion

In the following, the performances of the four models $\hat{f}_{\text{SSS,PD}}$, $\hat{f}_{\text{SSS,HSC}}$, $\hat{f}_{\text{MSS}}$ and $\hat{f}_{\text{CS}}$ are compared with respect to the accuracy and the number of parameters as a rough indicator for the computational effort. Let $n_{\overline{m}}$ be the number of samples where the output of $\hat{f}_{\text{SSS,PD}}$ before binarization is greater than $1-r$ or smaller than r, thus being the number of samples, where $\hat{f}_{\text{SSS,PD}}$ is certain. Let n_m be the number of all remaining samples, thus the number of samples, where $\hat{f}_{\text{SSS,HSC}}$ decides. $A_{\text{SSS,PD}\overline{m}}$ and $A_{\text{SSS,HSC}m}$ give the proportion of correctly classified samples, respectively. The accuracy A_{CS} of the cascaded system (CS) over all samples depending on the previously introduced threshold $r \in (0, 0.5)$ is thereby calculated by

$$A_{\text{CS}}(r) = \frac{n_{\overline{m}}(r) \cdot A_{\text{SSS,PD}\overline{m}}(r) + n_m(r) \cdot A_{\text{SSS,HSC}m}(r)}{n_{\overline{m}}(r) + n_m(r)}. \quad (7)$$

The accuracies $A_{\text{SSS,PD}}$, $A_{\text{SSS,HSC}}$, A_{MSS} and A_{CS} of the four models gained through 5-fold cross-validation are shown as black lines in Figure 7. Comparison of the accuracies of the four systems indicates that the SSS with the PD signals as input has the lowest. This is not surprising since in comparison to the SSS with the HSC images as input, only 13 samples of a time series are available at any time. The MSS having the highest accuracy can be explained by the fact that data of two sensors are analyzed simultaneously, and therefore a more holistic quality assessment is possible than with a single sensor. Thus, it can be inferred that there is information in the PD signal that is not captured by the HSC images.

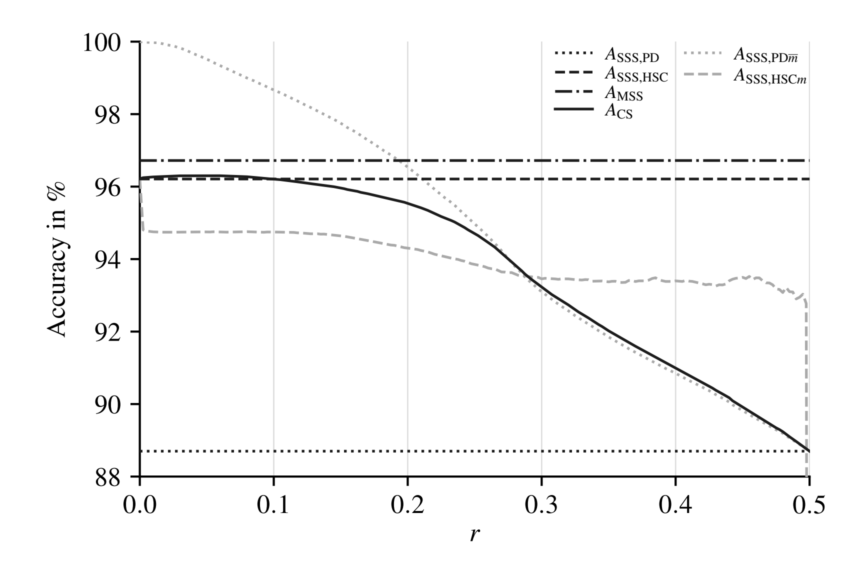


Figure 7: Accuracies. Black lines show the accuracies of the four models. The concrete accuracies of the models which are independent of r are $A_{\text{SSS,PD}} = 88.70\%$, $A_{\text{SSS,HSC}} = 96.21\%$ and $A_{\text{MSS}} = 96.72\%$. At $r = 0.053$, A_{CS} has its maximum of 96.31%. Gray lines indicate the accuracies $A_{\text{SSS,PD}\overline{m}}$ and $A_{\text{SSS,HSC}m}$ (cf. Equation 7).

In addition to the accuracies of the four models, the accuracies $A_{\text{SSS,PD}\overline{m}}$ and $A_{\text{SSS,HSC}m}$ are shown as gray lines in Figure 7. $A_{\text{SSS,PD}\overline{m}}$ indicates the accuracy, that $\hat{f}_{\text{SSS,PD}}$ decides right regarding samples where it is certain. If $r \to 0.5$ it converges to the accuracy $A_{\text{SSS,PD}}$ as all decisions are made by the SSS with PD samples as input. If $r \to 0$ it converges to 100%. This makes sense, since only those samples are considered for which the model is 100% sure. However, it is not self-evident as a 100% certain model can also decide wrongly. $A_{\text{SSS,HSC}m}$ indicates the accuracy of $\hat{f}_{\text{SSS,HSC}}$ for the samples, where the SSS of the PD is uncertain. The accuracy of these samples given by $\hat{f}_{\text{SSS,PD}}$ has to be below $A_{\text{SSS,PD}}$ as $A_{\text{SSS,PD}\overline{m}}$ always is above. The graph of $A_{\text{SSS,HSC}m}$ shows that $\hat{f}_{\text{SSS,HSC}}$ is already correct for over 93%, except for r near 0.5, of the samples for which $\hat{f}_{\text{SSS,PD}}$ is uncertain and has an accuracy less than $A_{\text{SSS,PD}} = 88.70\%$. If $r \to 0$, then $A_{\text{SSS,HSC}m} \to A_{\text{SSS,HSC}}$.

A_{CS} indicates the accuracy of the cascaded system. If $r \to 0.5$, the accuracy converges to the accuracy of the SSS with PD signals as input since that system

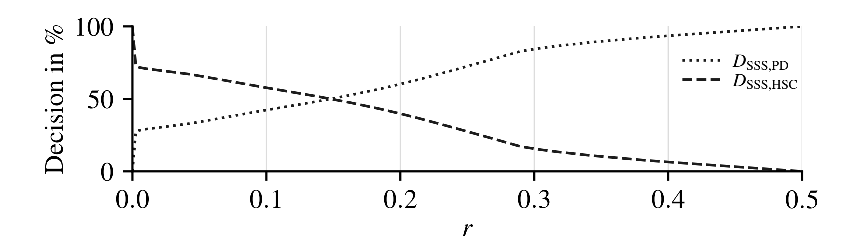


Figure 8: Final decisions. Shown are the percentages of the decisions in the cascaded system made by $\hat{f}_{SSS,PD}$ or $\hat{f}_{SSS,HSC}$ as a function of r.

makes all decisions; if $r \to 0$, A_{CS} converges to the accuracy of the SSS with HSC images as input, since then that system makes all decisions. With decreasing r, A_{CS} first follows $A_{SSS,PD\overline{m}}$ and then approaches $A_{SSS,HSC}$. Thereby, with $A_{SSS,PD\overline{m}}$ being the accuracy of samples using only the first classifier and $A_{SSS,HSCm}$ being the accuracy of the remaining samples resulting from the SSS of the HSC images, A_{CS} has to be in between of both (cf. Equation 7). At $r = 0.053$, A_{CS} has its maximum of 96.31%, which is greater than $A_{SSS,HSC}$.

Figure 8 shows what percentage of decisions in the CS are made by the respective system depending on r, where $r \to 0$ means that all decisions are made by the SSS with HSC images as input and $r \to 0.5$ that all decisions are made by the SSS with PD signals as input. For r near 0, an abrupt increase or decrease can be seen in the graphs. This is due to the distribution certainties of the SSS with the PD signals as input. This classifier is able to detect many anomalies with certainties near 100% as anomaly. For references, on the other hand, very few samples are classified with certainties near 100%. An explanation why the neural network learns in that way is the following: In the considered data some anomalies are clearly recognizable because they differ from the reference data. Hence, they are easily recognizable by the network. However, there are anomalies that look similar to reference data and are therefore very difficult to distinguish from references. Thus, the classifier does not often decide with certainties near 100% for a reference.

The number of parameters $P_{SSS,PD}$, $P_{SSS,HSC}$, P_{MSS} and P_{CS} in the inference of the systems, which is a rough indication for how long it takes the models to process data, are shown in Figure 9. Thereby, the MSS has the most parame-

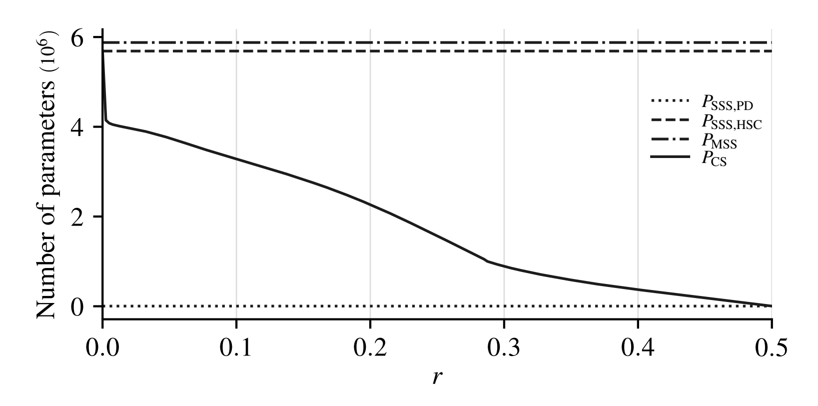


Figure 9: Number of parameters in the inference of the systems as measure for the computational effort. The concrete number of parameters are $P_{SSS,PD} = 3657$, $P_{SSS,HSC} = 5\,687\,233$ and $P_{MSS} = 5\,877\,961$.

ters, the SSS with the HSC images as input the second most and the SSS with the PD signals the least. The estimated computational effort of the CS results proportionally to how many samples are subjected to the first classifier only and how many are subjected to the first and second. If the decision is made by the SSS with the PD signals only, the estimated computational effort of the CS is equal to that of the SSS. With decreasing r the computational effort becomes larger. If $r \to 0$, than $P_{CS} \to P_{SSS,PD} + P_{SSS,HSC}$. If the CS has half of the final decisions made by the SSS with PD signals and the other half by the SSS with the HSC images, the estimated computational effort is reduced by 49.94% leading to an accuracy of 95.96%. Of interest is the region where the accuracy of the CS exceeds that of the SSS with the HSC images as input. At the point where A_{CS} is at its maximum and higher than $A_{SSS,HSC}$, the estimated computational effort is reduced by 34.46% compared to the SSS with HSC images.

6 General Cascaded System

The two-stage cascaded system proposed in this paper can be generalized. The structure of a general cascaded system (GCS) is shown in Figure 10. In contrast

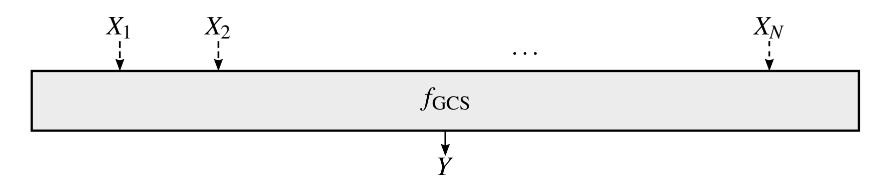


Figure 10: General cascaded system (GCS). The model f_{GCS} has $X_1, X_2, ..., X_N$ as inputs leading to a quality assessment Y.

to the two-staged cascaded system limited to two inputs, the GCS consists of N different inputs $X_1, X_2, ..., X_N$, which are (partially) processed together leading to a quality assessment Y. By that, the inputs do not necessarily have to come from different sensors, e.g. one sensor could provide several features as different inputs. Besides, possible inputs could also be metadata. The following structures of a cascaded system are possible, for example:

In contrast to the two-stage CS, the different input data $X_1, X_2, ..., X_N$ do not have to be divided into equal time periods. For example, considering the data used in this paper, input X_1 could make a decision for each entire weld seam. Further inputs can then work with smaller time periods in order to locate anomalies more precisely.

Unlike the proposed two-stage CS, which at uncertainty of a first classifier activates a second classifier, a GCS could exchange further information like learned features or certainties between classifiers. In doing so, the information flow does not have to be unidirectional. For example, speaking of a CS with two inputs, after making a first uncertain decision, the second classifier may return learned information to the first one in order to improve the first classifier's upcoming decision-making.

If there are several inputs, the system can decide which data $X_2, ..., X_N$ should be processed further depending on the results of the first input X_1. For example, if a first result supposes a specific anomaly among several, the next step evaluates exactly those inputs, that provide further information about that anomaly. The same principle can also be applied when choosing the initial inputs: Instead of using a fixed number of inputs in the first step, based on the decision of the previous step, initial inputs are selected. For example, if certain

anomalies are known to take a certain time, it could be useful to give attention to the inputs that can deal with the anomaly during that time period.

7 Summary and Outlook

Four different models for quality inspection of metal plates welds, namely two single-sensor system (SSS), a multi-sensor system (MSS) and a cascaded system (CS) have been considered. The CS presented in this paper is characterized by the fact that not all available data are analyzed to get the quality weld; only cleverly selected data are evaluated. The different models consisting of neural networks are compared in terms of their accuracy and estimated computational effort since fast but still precise quality inspection is needed in today's quality monitoring of welds. The data acquisition was done from scratch in the laboratory since no public data sets are available, whereat different anomalies such as spatters or gaps were inserted. Thereby, two common sensor methods for examining weld quality, namely a photodiode (PD) and a high-speed camera (HSC) have been used.

Among the four considered models, the SSS with PD signals as input has the fewest parameter but also the lowest accuracy. In contrast, the MSS has the highest accuracy but also the most parameters. If the CS has half of the final decisions made by the SSS with PD signals and the other half by the SSS with the HSC images, the estimated computational effort is reduced by almost 50% compared to the SSS with the HSC images as input. By that, the accuracy is only reduced by 0.25% from 96.21% to 95.96%.

Based on the CS, a generalized cascaded system (GCS) for quality inspection is proposed, which arbitrarily combines any number of sensors in order to get a quality assessment. Further work can deal with the possibilities that the GCS proposes, like the extension to more than two sensors or the incorporation of process knowledge. In addition to the used neural network architectures within the different models for feature extraction and classification, future work could include other architectures or methods based on classical machine learning. Furthermore, a more detailed analysis of the computational effort including other variables besides the number of parameters should be realized.

Moreover, the system can be extended to distinguish not only between anomaly and reference, but also between different anomalies.

References

[1] M. M. Atabaki, N. Yazdian, J. Ma, R. Kovacevic. "High power laser welding of think steel plates in a horizontal butt joint configuration". In: *Optics & Laser Technology*, vol. 83, pp. 1–12. 2016.

[2] S. Pang, X. Chen, J. Zhou, X. Shao and C. Wang. "3D transient multiphase model for keyhole, vapor plume, and weld pool dynamics in laser welding including the ambient pressure effect". In: *Optics and Lasers in Engineering*, vol. 74, pp. 47–58. 2015.

[3] Y. Zhang, F. Li, Z. Liang, Y. Ying, Q. Lin and H. Wei. "Correlation analysis of penetration based on keyhole and plasma plume in laser welding". In: *Journal of Materials Processing Technology*, vol. 256, pp. 1–12. 2018.

[4] C. Alippi, P. Braione, V. Piuri and F. Scotti. "A methodological approach to multisensor classification for innovative laser material processing units". In: *Proceedings of the 18th IEEE Instrumentation and Measurement Technology Conference (I2MTC)*, vol. 3, pp. 1762–1767. 2001.

[5] A. G. Paleocrassas and J. F. Tu. "Inherent instability investigation for low speed laser welding of aluminum using a single-mode fiber laser". In: *Journal of Materials Processing Technology* vol. 210, no. 10, pp. 1411–1418. 2010.

[6] A. Molino, M. Martina, F. Vacca, G. Masera, A. Terreno, G. Pasquettaz and G. D'Angelo. "FPGA implementation of time–frequency analysis algorithms for laser welding monitoring". In: *Microprocessors and Microsystems* vol. 33., no. 3, pp. 179–190. 2009.

[7] S. S. Rodil, R. A. Gómez, José M. Bernández, F. Rodríguez, L. J. Miguel and José R. Perán. “Laser welding defects detection in automotive industry based on radiation and spectroscopical measurements”. In: *The International Journal of Advanced Manufacturing Technology* vol. 49., pp. 133–145. 2010.

[8] F. Kong, J. Ma, B. Carlson and R. Kovacevic. “Real-time monitoring of laser welding of galvanized high strength steel in lap joint configuration”. In: *Optics & Laser Technology*, vol. 44, pp. 2186–2196. 2012.

[9] P. B. García-Allende, J. Mirapeix, O. M. Conde, A. Cobo and J. M. López-Higuera. “Spectral processing technique based on feature selection and artificial neural networks for arc-welding quality monitoring”. In: *NDT & E International* vol. 42., no. 1, pp. 56–63. 2009.

[10] M. Thornton, L. Han and M. Shergold. “Progress in NDT of resistance spot welding of aluminium using ultrasonic C-scan”. In: *NDT & E International*, vol. 48, pp. 30–38. 2012.

[11] K. Wasmer, T. Le-Quang, B. Meylan, F. Vakili-Farahani, M. P. Olbinado, A. Rack and S. A. Shevchik. “Laser processing quality monitoring by combining acoustic emission and machine learning: a high-speed X-ray imaging approach”. In: *Procedia CIRP*, vol. 74, pp. 654–658. 2018.

[12] S. Shevchik, T. Le-Quang, B. Meylan, F. V. Farahani, M. P. Olbinado, A. Rack, G. Masinelli, C. Leinenbach and K. Wasmer. “Supervised deep learning for real-time quality monitoring of laser welding with X-ray radiographic guidance”. In: *Sci. Rep.*, vol. 10, no. 1, p. 3389. 2020.

[13] M. Baader, A. Mayr, T. Raffin, J. Selzam, A. Kühl and J. Franke. “Potentials of Optical Coherence Tomography for Process Monitoring in Laser Welding of Hairpin Windings”. In: *11th International Electric Drives Production Conference (EDPC)*, pp. 1–10. 2021.

[14] S. Tsukamoto. “High speed imaging technique Part 2 – High speed imaging of power beam welding phenomena”. In: *Science and Technology of Welding and Joining*, vol. 16, no. 1, pp. 44–55. 2011.

[15] M. Jäger, S. Humbert and F. A. Hamprecht. “Sputter Tracking for the Automatic Monitoring of Industrial Laser-Welding Processes”. In: *IEEE Transactions on Industrial Electronics*, vol. 55, pp. 2177–2184. 2008.

[16] M. Jäger and F. A. Hamprecht. “Principal Component Imagery for the Quality Monitoring of Dynamic Laser Welding Processes”. In: *IEEE Transactions on Industrial Electronics*, vol. 56, pp. 1307–1313. 2009.

[17] J. Vater, M. Pollach, C. Lenz, D. Winkle and A. Knoll. “Quality Control and Fault Classification of Laser Welded Hairpins in Electrical Motors”. In: *28th European Signal Processing Conference (EUSIPCO)*, pp. 1377–1381. 2017.

[18] B. Zhou, T. Pychynski, M. Reischl, E. Kharlamov and R. Mikut “Machine learning with domain knowledge for predictive quality monitoring in resistance spot welding”. In: *Journal of Intelligent Manufacturing*, vol. 33, no. 4, pp. 1139–1163. 2022.

[19] E. B. Schwarz, F. Bleier, F. Guenter, R. Mikut and J. P. Bergmann “Improving process monitoring of ultrasonic metal welding using classical machine learning methods and process-informed time series evaluation”. In: *Journal of Manufacturing Processes*, vol. 77, pp. 54–62. 2022.

[20] T. S. Yun, K. J. Sim and H. J. Kim. “Support vector machine-based inspection of solder joints using circular illumination”. In: *Electronics Letters* vol. 36, no. 11, p. 1. 2000.

[21] X. Hongwei, Z. Xianmin, K. Yongcong and O. Gaofei. “Solder joint inspection method for chip component using improved AdaBoost and decision tree”. In: *IEEE Transactions on Components, Packaging and Manufacturing Technology* vol. 1, no. 12, pp. 2018-2027. 2011.

[22] H. Wu. “Solder joint defect classification based on ensemble learning”. In: *Soldering & Surface Mount technology*, vol. 29, no. 3, pp. 164–170. 2017.

[23] H. Wu, X. Zhang, H. Xie, Y. Kuang and G. Ouyang. “Classification of Solder Joint Using Feature Selection Based on Bayes and Support Vector Machine”. In: *IEEE Transactions on Components, Packaging and Manufacturing Technology*, vol. 3, no. 3, pp. 516–522. 2013.

[24] A. Krizhevsky, I. Sutskever and G. E. Hinton. “Imagenet Classification with Deep Convolutional Neural Networks”. In: *Communications of the ACM* vol. 60, no. 6, pp. 84–90. 2017.

[25] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens and Z. Wojna. “Rethinking the Inception Architecture for Computer Vision”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. pp. 2818–2826. 2016.

[26] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le and H. Adam. “Searching for MobileNetV3”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* pp. 1314–1324. 2017.

[27] Y. Yang, L. Pan, J. Ma, R. Yang, Y. Zhu, Y. Yang and L. Zhang. “A High-Performance Deep Learning Algorithm for the Automated Optical Inspection of Laser Welding”. In: *Applied Science* vol. 10, no. 3. 2020.

[28] P. Stritt, M. Boley, A. Heider, F. Fetzer, M. Jarwitz, D. Weller, R. Weber, P. Berger and T. Graf. “Comprehensive process monitoring for laser welding process optimization”. In: *Proc. SPIE 9741, High-Power Laser Materials Processing: Lasers, Beam Delivery, Diagnostics, and Applications V, 97410Q*. 2016.

[29] Y. Zhang, D. You, X. Gao, N. Zhang and P. P. Gao. “Welding defects detection based on deep learning with multiple optical sensors during disk laser welding of thick plates”. In: *Journal of Manufacturing Systems*, vol. 51, pp. 87–94. 2019.

[30] Y. Yang, R. Yang, L. Pan, J. Ma, Y. Zhu, T. Diao and L. Zhang “Real-time monitoring of high-power disk laser welding statuses based on deep learning framework”. In: *Journal of Intelligent Manufacturing*, vol. 31, pp. 799–814. 2020.

[31] T.-H. Kim, T.-H. Cho, Y. S. Moon, S. H. Park. “Visual inspection system for the classification of solder joints”. In: *Pattern Recognition*, vol. 32, no. 4, pp. 565–575. 1999.

[32] S.-C. Lin, C. H. Chou, and C.-H. Su. “A development of visual inspection system for surface mounted devices on printed circuit board”. In: *IECON 2007-33rd Annual Conference of the IEEE Industrial Electronics Society*, pp. 2440–2445. 2007.

[33] K. Y. Chan, K. F. C. Yiu, H.-K. Lam and B. W. Wong. “Ball bonding inspections using a conjoint framework with machine learning and human judgement”. In: *Applied Soft Computing* vol. 102, no. 107115. 2021.

[34] Z. Wang, W. Yan and T. Oates. “Time Series Classification from Scratch with Deep Neural Networks: A Strong Baseline”. In: *International Joint Conference on Neural Networks (IJCNN)*. 2017.

[35] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam. “MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications”. In: *arXiv:1704.04861*. 2017.

Predicting the Compressive Strength of Concrete up to 28 Days-Ahead: Comparison of 16 Machine Learning Algorithms on Benchmark Datasets

Farzad Rezazadeh[1], Andreas Kroll[2]

[1,2]Department of Measurement and Control, University of Kassel
Moenchebergstr. 7, 34125 Kassel
E-Mail: {farzad.rezazadeh, andreas.kroll}@mrt.uni-kassel.de

Abstract

Concrete is the most important and widely consumed construction material. Concrete parts are produced by a mixing process, followed by casting and a certain curing time. To assess the quality of concrete, its compressive strength is usually measured (typically after 28 days curing time). Several factors affect the compressive strength of concrete, including environmental factors, the type, quality, and quantity of the constituents, the order of the mixing process, and the curing conditions. Due to the multitude of factors effecting compressive strength and partially known chemical reactions during mixing and curing, in this contribution, data-driven methods are used to model the behavior of the concrete production process. Three different benchmark datasets from the concrete manufacturing field are used for the modeling procedure. 16 typical learning algorithms were selected based on their simplicity and their performance in predicting compressive strength. The results show that 1) repeated cross-validation is more reliable than repeated hold-out in this configuration, 2) the interaction and power terms (2nd order) of the inputs have a positive effect on model prediction, 3) the kernel type of the models is of crucial importance, and 4) gradient boosting and kernel ridge are the most appropriate models for predicting compressive strength.

1 Introduction

Conventional concrete (CC) consists of the basic materials Portland cement, fine and coarse aggregate, and water. To achieve higher compressive strength (CS) and better workability of the concrete, the basic materials are supplemented and mixed with additives such as fly ash, silica fume, blast furnace slag, and also superplasticizer in various recipes [1]. Depending on the additives and the type of mixing, high-performance concrete (HPC) or ultra-high performance concrete (UHPC) is produced, as indicated in Table 1. Figure 1 shows an overview of the four steps of the concrete production process. In the first step, the raw materials chosen according to the recipe that are subject to environmental conditions (temperature, humidity), are poured into the mixer according to the specific instructions of the recipe. In the second step, the mixer is first set to a certain speed and duration based on the specific instructions recipe. The result of the second step is fresh concrete. The important factors in fresh concrete are the temperature, electrical conductivity and the amount of air contained. The results of the slump and flow tests can be used as additional describing factors. Lastly, after the curing period (storage of the fresh concrete under certain conditions to make it hard), the final concrete production is prepared. If the compressive strength after curing (lasts typically for 28 days) could be predicted during production (ideally already during the mixing process) off-spec product could be foreseen, correcting action taken and the delivery of off-spec parts be avoided.

1.1 State-of-the-Art Regarding Considered Model Inputs/Features

Rajeshwari et al. [3] attempt to analyze the use of different amounts of fly ash in concrete formulations and conclude that using a large amount of fly ash not only improves the physical properties of concrete, but also reduces greenhouse gas production and is also cost effective. In studying the effects of environmental conditions (during curing), and cement types on the compressive strength of concrete, Farzampour [4] concluded that the temperature and the cement-to-water ratio in extreme weather conditions, are highly critical.

Table 1: CC, HPC, and UHPC can be distinguished by their important characteristics [2].

Concrete Unit	CS MPa	Water/binder %	Workability mm	Cement kg/m^3
CC	20-50	0.45-0.65	-	260-380
HPC	50-100	<0.4	455-810	400-700
UHPC	100<	0.2-0.3	260<	800-1000

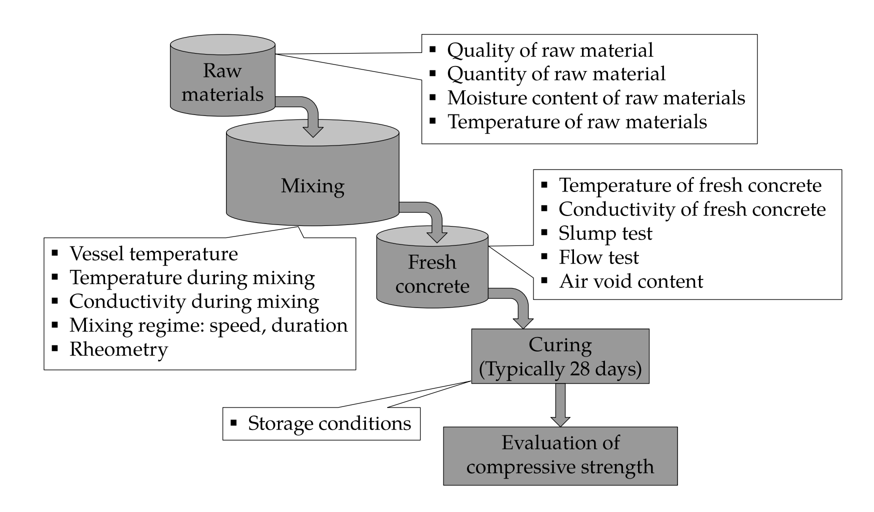


Figure 1: Process of concrete production and what to consider at each step.

Non-destructive tests such as electrical conductivity [5] and ultrasonic pulse velocity [6] measurement to assess the concrete quality are also used because of their low cost.

1.2 State-of-the-Art Regarding Model Types

Modeling of the concrete production process in order to estimate the concrete quality (usually the compressive strength after 28 days) is generally divided into two categories: traditional modeling (based on empirical relationships) and machine learning. In case of the traditional modeling method, according

to Abram's law [7], an empirical relationship based on the ratio of water to cement is used to estimate the compressive strength ($f = b_1/(b_2^{w/c})$, where f is compressive strength after 28 days curing time and b_1, b_2 are empirical constants). In order to complete Abram's law and improve the accuracy of the estimation, Zain et al. [8] proposed a multiple linear regression,

$$f = b_0 + b_1 \frac{w}{c} + b_2 a_1 + b_3 a_2 + c, \tag{1}$$

where f is the compressive strength (after 28 days curing time), w is the water volume, c is the amount of cement, a_1 is the amount of coarse aggregate, a_2 is the amount of fine aggregate, and b_0, b_1, b_2, and b_3 are empirical constants. On the other hand, Zhu et al. [9] have attempted to estimate the compressive strength *f(t)* based only on the curing time t,

$$f(t) = b_0 + b_1 t + b_2 t^2 + b_3 t^3. \tag{2}$$

Due to the wide spectrum of effects on compressive strength and the complex and incompletely known chemical reactions during mixing and curing, as well as the inability of classical modeling methods to address a broad range of influencing factors, data-driven methods for modeling the behavior of the concrete production process have prevailed. Ling et al. [10] compared support vector machine (SVM), artificial neural network (ANN), and decision tree (DT) to study the effect of environmental factors on the compressive strength and concluded that the SVM performed better than other methods. On the other hand, Hoang et al. [11] show that the nonparametric and stochastic algorithm Gaussian Process Regression (GP) has a better ability to estimate compressive strength than ANN and SVM. Compared to the above methods, the best estimates for the compressive strength of concrete were obtained using ensemble learning regression [12].

1.3 Scope of Present Work

In this contribution, the performance of models with different structures - from linear to nonlinear, from parametric to nonparametric, and from single model

to ensemble - is compared for three concrete benchmark datasets that have different dimensions and a different number of data. At first, the models are trained using two alternative methods: repeated cross-validation and repeated hold-out [13]. The performance variation of the models is assessed based on 20 different initializations for the cross-validation and hold-out methods as well as for each model. The effect of Bayesian optimization is examined on the hyperparameters of the models. The effects of interaction and power terms (2nd and 3rd order) of the inputs are also investigated. For using of the models in online applications, the processing speed of each model is evaluated along with the memory required to estimate the output.

2 Benchmark Datasets Characterization

Concrete production in the laboratory requires purchasing expensive raw materials and handling the manufacturing process. Typically, the production of only one data takes 28 days. This means that the creation of a small dataset with, for example, only 100 data points can take more than half a year dependent on the lab capacity. On the other hand, in the concrete industry, only a few recipes are applied, i.e. there is only little variation in the production process, so that the resulting data may not be very informative. The data available in the literature are usually generated in laboratories and, for simplicity, are based only on changing some constituents, with the environmental variables assumed to be constant [11]. Alternatively, the recipe and the type, quality and quantity of constituents are assumed to be constant and only some environmental factors (usually one or two) are changed [4].

In this paper, three different benchmark datasets from the concrete manufacturing field are used for the modeling procedure. The first dataset (Bdata) [14], was collected from 17 literature sources (laboratory data) and includes 1030 data points. The second dataset (Sdata) [15] with 103 data points and the third dataset (XSdata) [6] with 84 data points originate also from laboratory experiments. Table 2 lists characteristic values for each dataset. As shown, seven factors are identical in Bdata and Sdata. Comparing the minimum and maximum values of the ingredients of Bdata and Sdata, it can be seen that

Table 2: Characterization of the concrete benchmark datasets: Mean, median, standard deviation (STD), minimum (MIN), and maximum (MAX) values of the individual factors.

(a) Bdata (N = 1030)

Ingredient	Unit	Mean	Median	STD	MIN	MAX
Cement	kg/m^3	281.16	272.90	104.50	102	540
Blast furnace slag	kg/m^3	73.89	22	86.27	0	359.40
Fly ash	kg/m^3	54.18	0	63.99	0	200.10
Water	kg/m^3	181.56	185	21.35	121.80	247
Superplasticizer	kg/m^3	6.20	6.40	5.9	0	32.20
Coarse aggregate	kg/m^3	972.91	968	77.75	801	1145
Fine aggregate	kg/m^3	773.58	779.50	80.17	594	992.60
Age	day	45.66	28	63.16	1	365
Compressive strength	MPa	35.81	34.44	16.70	2.33	82.6

(b) Sdata (N = 103): The slump and flow tests assess the fluidity (looseness or stiffness) and the consistency of the fresh concrete, respectively.

Ingredient	Unit	Mean	Median	STD	MIN	MAX
Cement	kg/m^3	229.89	248	78.87	137	374
Blast furnace slag	kg/m^3	77.97	100	60.46	0	193
Fly ash	kg/m^3	149.01	164	85.41	0	260
Water	kg/m^3	197.16	196	20.20	160	240
Superplasticizer	kg/m^3	8.53	8	2.80	4.40	19
Coarse aggregate	kg/m^3	883.97	879	88.39	708	1050
Fine aggregate	kg/m^3	739.60	742.70	63.34	640.60	902
Slump test	cm	18.04	21.50	8.75	0	29
Flow test	cm	49.61	54	17.56	20	78
Compressive strength	MPa	36.03	35.52	7.83	17.19	58.53

(c) XSdata (N = 84)

Ingredient	Unit	Mean	Median	STD	MIN	MAX
Blast furnace slag/binder	%	0.30	0.30	0.22	0	0.60
Ultrasonic pulse velocity	km/s	4.14	4.18	0.39	3.16	4.81
Age	day	73.42	56	65.01	3	180
Compressive strength	MPa	30.45	30.50	12.94	6.32	54.14

Bdata generally uses a larger range to create each sample. In general, Bdata has a high dimenson with large data points, Sdata has almost the same high dimenson as Bdata but with significantly fewer data points, and XSdata has low dimensions and also less data points.

3 Used Data-Driven Modeling Methods

To model concrete manufacturing process, linear vs. nonlinear single models and nonlinear ensemble models are investigated. In the ensemble structure, instead of using a single model to predict the system behavior, a combined structure with multiple algorithms is used to explore the data from different angles and achieve a better understanding of the patterns by combining their results into a final prediction [12]. In this contribution, only two ensemble structures (bagging and boosting) are considered with decision trees as the base learners. All used models originate from the Sklearn [16], Xgboost [17], and LightGBM [18] Python's libraries. A PC with Intel(R) Core(TM) i9-10900X CPU and 64.0 GB RAM is used. Tables 4 and 5 give an overview of the used models with the considered hyperparameters (HPs).

Bayesian optimization, from the Skopt [16, 32] library in Python, is used to determine the optimal hyperparameters. Bayesian optimization is a non-derivative and fast optimization technique that searches to find the global minimum. Bayesian optimization uses an optimization procedure to create a probability model, also known as a surrogate model of the objective function, and selects a hyperparameter in each iteration based on the value of an acquisition function in the previous iteration for the probability function [33]. The best hyperparameter is selected based on the best value of the acquisition function during the whole procedure.

To achieve a more realistic and also generalized performance measure for each algorithm, 20 times repeated hold-out and repeated cross-validation [13] are applied to split each of the three datasets into training and test datasets. The same dataset splits are applied for each model training. In each repetition, a random initialization is used. Finally, the average performance of each algorithm in the 20 runs is calculated (Algorithm 1 for the repeated hold-out).

Table 4: Overview of linear and nonlinear single models/algorithms

	Model	Ab.	Description	HP	Re.
Linear models	Linear Regression	LI	Model with tunable coefficients minimizing the difference between the estimated and true outputs	None	[12]
	Lasso	LS	Model that provides a sparse remedy based on the L1 penalty	α1 (L1 coefficient)	[19]
	Ridge	RG	Model that provides a sparse remedy based on the L2 penalty	α2 (L2 coefficient)	[20]
	Kernel Ridge	KR	Combination of RG and kernel trick	α2, kernel	[21]
	Elastic Net	EN	Model that provides a sparse remedy based on the L1 and L2 penalties	α1, α2	[22]
Nonlinear single models	Support Vector Machine	SV	Finding the line/hyperplane based on minimizing the distance between predicted and true values within highest confidence margin	α2, kernel, polynomial kernel degree	[23]
	K-nearest neighbor	KN	Output based on the average of the k-values of the nearest neighbor points	Number and distance type of neighbors	[24]
	Gaussian Process Regression	GP	Non-parametric Bayesian modeling	Kernel	[25]
	Decision Tree	DT	Non-parametric model for estimating output based on straightforward decision rules	Depth of tree	[26]

4 Results

It should be noted that after investigating the correlation between the factors of each dataset, it is found that only in Sdata there is a strong correlation just between slump and flow (90 %). So slump is chosen as an input for modeling, since the model performances are almost the same when training the models with and without flow.

Table 5: Overview of nonlinear ensemble models/algorithms

	Model	Ab.	Description	HP	Re.
Nonlinear ensemble models	Random Forest	RF	Model based on the different DTs using bootstrap replicas	Number of trees	[27]
	Extra Tree	ET	Model based on the different DTs using whole dataset	Number of trees	[28]
	AdaBoost	AB	Prediction based on decision stumps grounded on DT with only one node and two leaves	Number of trees, learning rate	[29]
	Gradient Boosting	GB	Based on DT and gradient descent optimization, starting from a single leaf as opposed to AB (AB starts from a stump)	Number of trees, learning rate	[30]
	Stochastic Gradient Boosting	XB	Based on GB and regularization principle to avoid overfitting. It increases the leaves in the horizontal direction	Number of trees, depth of tree, $\alpha 1$, $\alpha 2$	[17]
	Light Gradient Boosting	LB	LB increases the leaves in the vertical direction, which leads to a better and also faster performance than XB	Number of trees, learning rate	[18]
	Histogram Gradient Boosting	HB	Based on GB with a preprocessing technique through discretization to group data	Depth of tree, learning rate, $\alpha 2$	[31]

The results of the 20 repetitions of hold-out and cross-validation are shown in Figures 2, 3, and 4 for Bdata, Sdata, and XSdata, respectively. The abbreviations of the model types are defined in Tables 4 and 5. The evaluation criterion used here is R^2 because it is simple, standardized, and the most widely used criterion in predicting compressive strength:

$$R^2(y,\hat{y}) = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y}_i)^2} \ , \tag{3}$$

where $\bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i$ and $\hat{y}$ is predicted y. With the hold-out method, the performance of the model depends on the choice of the data for training and testing. This dependence is evident in the modeling results of all three datasets. For 20

Algorithm 1.: Repeated hold-out procedure

```
model ← one choice from 16 available algorithms
dataset ← one choice from Bdata, Sdata, or XSdata
results ← []
splitSize ← 10 % for test data (90 % for training data)

  for i ← 0:20 (20 times hold−out)
      seed ← initialize with random values
      Train, Test ← Splitting dataset based on seed
      optimHypers ← []
      for Fold ← 10 (Bayesian optimization loop)
          train, test ← Splitting Train based on seed
          optimHyper ← optimal hyperparameters (model)
          optimHypers ← [optimHypers, optimHyper]
      end
      OHP ← best hyperparameters from optimHypers
      trainedModel ← fit model with Train (seed, OHP)
      result ← evaluate trainedModel with Test
      results ← [results, result]
  end
  RESULT ← mean of results
  return RESULT
```

repetitions of the hold-out procedure, the performance varies in a large range. For example, LI performance varies from 45 % to 66 % for Bdata, from 75 % to 94 % for Sdata, and from 60 % to 93 % for XSdata. In reviewing the literature estimating the concrete behavior, in most of the papers the results are obtained using the (one-time) hold-out method, without any other initial randomization of the model training. Such results are not reliable and reproducible, since the best performance of the model could be selected by chance. For example, in this contribution, SV performance for the repeated hold-out (20 runs) is equal to $R^2_{max} = 73.94$ %, $R^2_{mean} = 63.76$ %, and $R^2_{min} = 53.16$ % for Bdata and $R^2_{max} = 94.28$ %, $R^2_{mean} = 86.08$ %, and $R^2_{min} = 71.49$ % for Sdata. However, in [34] the reported SV performance for Bdata is $R^2 = 73.98$ %, and in [35] for Sdata is $R^2 = 85.50$ % (only one hold-out). In such a case, the performance of the model with real data may show a large deviation. To reduce the impact of this problem, the repeated hold-out method is used. However, the disadvantage of the repeated hold-out method is the possibility that some data never appear

in the training or in the testing process. Cross-validation is an alternative, but to obtain results with higher confidence, repeated cross-validation is recommended [13]. In Figures 2, 3, and 4, the performance of each model based on repeated cross-validation varies in a smaller range and also generally has a smaller median than that of the hold-out method. This is because in each boxplot, each point of 20 runs of cross-validation represents an average of 10 training-test procedures (10-fold corss-validation), but in contrast, in each boxplot of the hold-out method, each point of 20 runs represents only one training-test procedure.

As shown in Figures 2, 3, and 4, the performance of the linear models is quite weak for Bdata, where the dimension of the input space and the number of data points are high, and also in XSdata, where both the dimension and the number of data are low. But these linear models have shown good performance for Sdata, which has a high dimension like Bdata and a small number of data like XSdata. As a result, KR achieved the best performance for Sdata.

The ensemble models used are all developed based on decision trees, and in this context, the ensemble models perform better on average than their base learner. An excellent performance of ensemble models occurs for Bdata, but these models have not shown good results for Sdata, especially when compared to linear models. The reason is that these models require a large amount of data for training. This data requirement also depends on the dimension of the input space, so ensemble models perform much better in XSdata, which has almost the same amount of data but a much smaller dimension of the input space than Sdata. In general, the biggest challenge is the high dimension and small number of data in Sdata compared to the other two datasets. In such a situation, ensemble models based on decision trees and other conventional nonlinear models are not a good choice. On the other hand, GP adapts well to the small amount of data. By applying the kernel trick to the linear model (KR), i.e., by better simulating the nonlinear hidden pattern of the data in the model, better results can be obtained for Sdata than with other models.

To examine the effect of Bayesian optimization, the entire process of hold-out and cross-validation (with 20 runs) is performed both with and without the hyperparameter optimization loop. The average of the 20 runs of each model

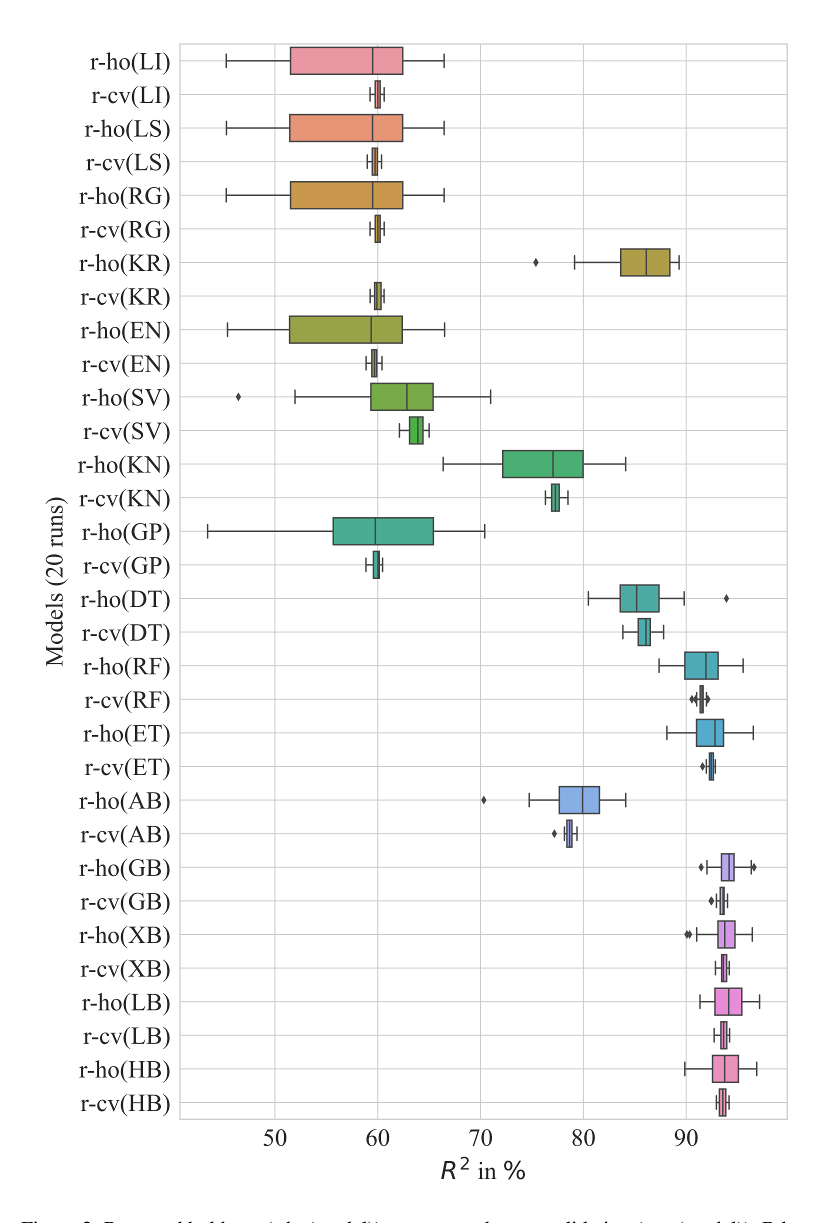


Figure 2: Repeated hold-out (r-ho(model)) vs. repeated cross-validation (r-cv(model)), Bdata

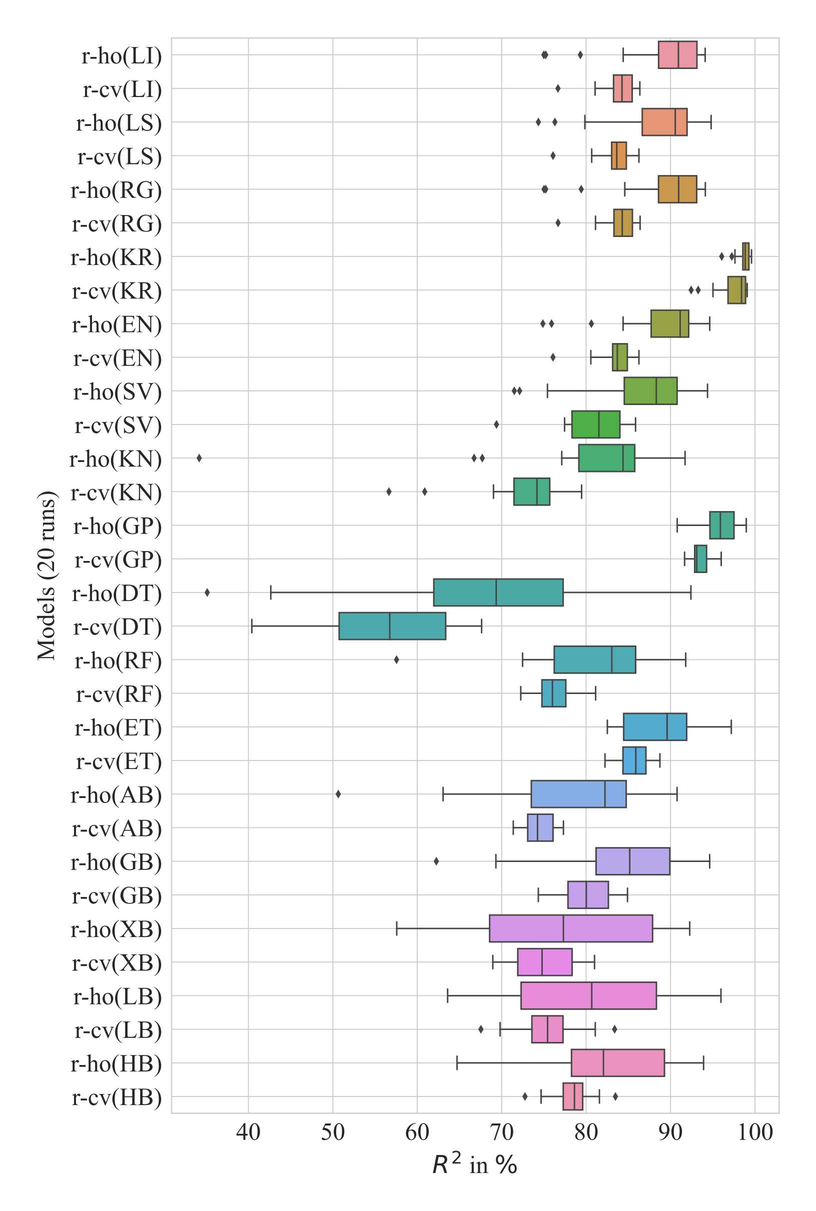


Figure 3: Repeated hold-out (r-ho(model)) vs. repeated cross-validation (r-cv(model)), Sdata

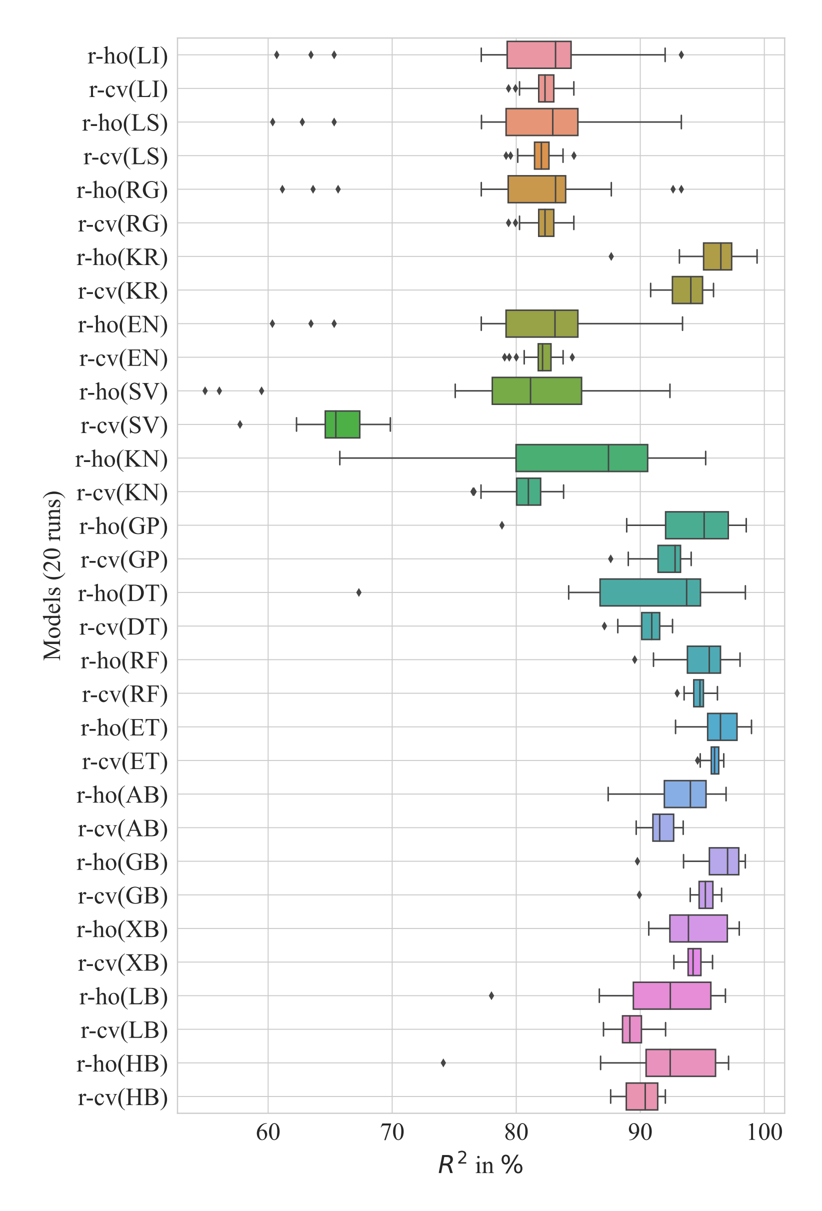


Figure 4: Repeated hold-out (r-ho(model)) vs. repeated cross-validation (r-cv(model)), XSdata

in all three datasets is presented in Table 6. In general, there is not much difference between the time required to train a model without the optimization loop in the three datasets. For example, KN takes 0.5, 0.4, and 0.4 seconds as training time for Bdata, Sdata, and XSdata; AB takes 0.2, 0.2, and 0.1; and HB needs 0.5, 0.4, and 0.4 seconds, respectively. But there is a big difference in the training time needed, if KN is utilized in the optimization loop. For example, KN needs 49 seconds for Bdata, 41 seconds for Sdata and 42 seconds for XSdata. AB takes 80, 61, 60 and HB takes 151, 122, and 113 seconds, respectively. The Bayesian optimization loop can increase the training time by more than 100 times. Now the question arises whether such an optimization loop is necessary in the modeling process. From Table 6, it can be concluded that Bayesian optimization improves the models where the kernel types and the number of neighbors are considered in the optimization loop (see the performance of KR, SV, GP, and KN). On the other hand, the linear models (RG and EN) where the regularization coefficient is a hyperparameter, or even the models based on the decision trees, do not differ much when the optimization is used.

The next step is to investigate the effect of the polynomial forms (interaction and power terms with degree two and three) of the input space on the performance of the models. Transferring the data into quadratic polynomial terms (into the interaction and also the power terms) resulted in a significant improvement in the average accuracy of the linear models. For example, the average LI performance in Bdata increased from 57.34 % to 68.31 % and in Sdata from 88.98 % to 97.23 %. In contrast, for the nonlinear models, this transfer of input space had a slightly positive effect on Bdata (KNN from 75.91 % to 76.80 %) and even a negative effect on Sdata (AB from 78.37 % to 67.59 %). These engineered nonlinear patterns (polynomial terms of 2nd degree) help linear models to recognize the nonlinear behavior of the data. On the other hand, nonlinear models do not require such feature engineering, and using higher order input terms introduces more complexity in the search space for models and can weaken the performance. Transferring the inputs to polynomial space with degree 3 had a negative effect on the performance of the models in all three datasets.

Table 6: Mean values of 20 runs of repeated hold-out (r-ho(model)) and repeated cross-validation (r-cv(model)), with (R^2 in %) and without (R_0^2 in %) Bayesian optimization.

	Bdata		Sdata		XSdata	
Model	R^2	R_0^2	R^2	R_0^2	R^2	R_0^2
r-ho(LI)	57.34	-	88.98	-	80.76	-
r-cv(LI)	**59.97**	-	**83.92**	-	**82.19**	-
r-ho(LS)	57.32	57.47	88.39	88.80	80.73	77.65
r-cv(LS)	**59.71**	59.82	**83.39**	82.64	**81.95**	73.56
r-ho(RG)	57.34	57.52	89	88.63	80.85	83.94
r-cv(RG)	**59.97**	59.84	**83.96**	82.91	**82.19**	81.05
r-ho(KR)	85.03	57.54	98.71	88.41	96.03	59.94
r-cv(KR)	**59.94**	59.90	**97.49**	82.78	**93.77**	51.80
r-ho(EN)	57.36	57.50	88.73	88.36	80.83	42.54
r-cv(EN)	**59.65**	59.84	**83.50**	82.95	**82.04**	36.29
r-ho(SV)	61.82	24.19	86.08	2.57	78.87	33.19
r-cv(SV)	**63.76**	25.01	**80.98**	2.03	**65.65**	23.63
r-ho(KN)	75.91	68.97	79.98	74.46	84.75	81.88
r-cv(KN)	**77.31**	71.19	**72.75**	59.49	**80.58**	77.81
r-ho(GP)	58.95	3.26	95.69	2.18	93.95	88.63
r-cv(GP)	**59.87**	3.14	**93.45**	3.12	**92.13**	87.97
r-ho(DT)	85.75	85.44	68.33	57.33	90.56	92.96
r-cv(DT)	**85.91**	85.76	**55.93**	56.29	**90.63**	89.54
r-ho(RF)	91.60	91.73	81.13	79.05	95	96.28
r-cv(RF)	**91.48**	91.62	**76.15**	75.54	**94.70**	94.42
r-ho(ET)	92.55	92.45	88.89	87.82	96.43	97.02
r-cv(ET)	**92.42**	92.53	**85.72**	85.22	**95.93**	95.82
r-ho(AB)	79.39	78.43	78.37	78.55	93.27	94.94
r-cv(AB)	**78.60**	78.58	**74.43**	72.71	**91.69**	92.30
r-ho(GB)	94.10	90.06	84.56	83.62	96.32	96.87
r-cv(GB)	**93.45**	90.49	**79.87**	80.52	**94.99**	95.07
r-ho(XB)	93.68	93.50	77.44	78.55	94.37	95.36
r-cv(XB)	**93.63**	93.46	**75.23**	74.57	**94.34**	94.18
r-ho(LB)	94	93.41	80.31	83.34	91.71	91.96
r-cv(LB)	**93.64**	93.31	**75.60**	76.24	**89.23**	89.59
r-ho(HB)	93.74	93.40	82.44	82.98	91.93	92.42
r-cv(HB)	**93.53**	93.36	**78.27**	77.18	**90.14**	90.51

It should also be kept in mind that transferring the inputs to the higher order polynomial space significantly increases the time required in the optimization loop, in the training process, and in the testing phase, so a trade-off between time and accuracy should be considered when applying such techniques.

Table 7 illustrates three important attributes for the evaluation of each model under online application, i.e., performance (R^2), prediction time (P_t), and memory requirement for prediction (P_m). For Bdata, GB model with $R^2 = 93.45$ %, $P_t = 1.40$ ms, and $P_m = 18114$ KiB performs better than the other models in all three factors. GB is followed by the LB models with high performance ($R^2 = 93.64$ %) and acceptable memory requirement ($P_m = 24082$ KiB) but high prediction time ($P_t = 16.71$ ms), and AB ($R^2 = 78.60$ %, $P_t = 3.59$ ms, $P_m = 216247$ KiB), and KN ($R^2 = 77.31$ %, $P_t = 2.50$ ms, $P_m = 67315$ KiB) with acceptable prediction time and memory requirement but lower performance. In Sdata, KR with $R^2 = 97.49$ %, $P_t = 17.65$ ms and $P_m = 10351$ KiB are able to be the best choice, but on the other hand, GB with $R^2 = 79.87$ %, $P_t = 0.62$ ms, and $P_m = 3440$ KiB needs less speed and memory than KR, but its performance is weaker. Finally, for XSdata, GB with $R^2 = 94.99$ %, $P_t = 0.62$ ms, and $P_m = 3044$ KiB is the best choice for online application.

5 Conclusions

In this study, 16 data-driven modeling algorithms of linear and nonlinear, parametric and nonparametric type, and ensembles are investigated for predicting compressive strength of concrete. For this, three datasets with different number of dimensions and different number of data are used. Based on the results of the repeated hold-out and repeated cross-validation methods, it is recommended to use the repeated cross-validation in the training process when the required high computational power is available. The results vary less, which means that it provides a more reliable assessment of the model performance. It is also recommended to use Bayesian optimization only for models with kernels (KR, SV, and GP) and for KN. In cases where the regularization coefficient or the number and depth of the trees are hyperparameters for the model, Bayesian

Table 7: Performance of the models for the online application based on R^2 (in %) vs. prediction time (P_t in ms) vs. memory consumed for prediction (P_m in KiB).

	Bdata			Sdata			XSdata		
Model	R^2	P_t	P_m	R^2	P_t	P_m	R^2	P_t	P_m
LI	59.97	16.56	14969	83.92	22.81	4200	82.19	17.18	4596
LS	59.71	18.90	14958	83.39	23.90	3984	81.95	23.43	4036
RG	59.97	6.09	14895	83.96	21.09	3488	82.19	22.34	3092
KR	59.94	29.53	772124	97.49	17.65	10351	93.77	19.53	8172
EN	59.65	10	14963	83.50	24.37	3552	82.04	19.21	3156
SV	77.31	2.50	67315	80.98	436.09	71723	65.65	435.31	70731
KN	59.87	16.25	996430	72.75	5.46	34320	80.58	5.62	24136
GP	91.48	69.53	106916	93.45	50.78	99583	92.13	45.15	101366
DT	63.76	15.46	15089	55.93	21.71	4648	90.63	18.28	4252
RF	85.91	25.46	18102	76.15	13.75	3432	94.70	17.34	3036
ET	92.42	28.59	94573	85.72	56.56	84839	95.93	55.78	87991
AB	78.60	3.59	216247	74.43	2.65	24079	91.69	2.34	22671
GB	93.45	1.40	18114	79.87	0.62	3440	94.99	0.62	3044
XB	93.63	51.71	17183	75.23	55	17711	94.34	50.93	17711
LB	93.64	16.71	24082	75.60	32.50	12010	89.23	36.87	10578
HB	93.53	36.25	57485	78.27	58.28	49041	90.14	54.37	49585

optimization is not required and the model can be used directly as a tool with default values for the hyperparameters. The possible explanation are that the default values of the hyperparameters of the models used are generally optimal or that the types of the surrogates (Gaussian process), the acquisition functions (Expected improvement) and also the number of iterations (50) used for Bayesian optimization are not the best choices and other selections should be used for such cases. The consideration of the quadratic polynomial terms is recommended, especially in case of small datasets and for linear models (it is not recommended for nonlinear models). This concept of the polynomial terms can be developed specifically for the ensemble methods based on other base learners (not only DTs). Finally, due to the high performance and speed of GB for Bdata and XSdata, and the high performance and acceptable speed of KR for Sdata (if only limited resources are available, GB is better choice

than KR for Sdata), these models are recommended as superior models for the considered application.

Acknowledgment

This project (HA project no. 1011/21-13) is financed with funds of LOEWE – Landes-Offensive zur Entwicklung Wissenschaftlich-ökonomischer Exzellenz, Förderlinie 3: KMU-Verbundvorhaben (State Offensive for the Development of Scientific and Economic Excellence).





References

[1] D. Dutta and S. V. Barai. “Prediction of compressive strength of concrete: machine learning approaches”. In: *In Recent Advances in Structural Engineering* 1. pp. 503-513. 2019.

[2] M. T. Marvila, A. R. G. de Azevedo, P. R. de Matos, S. N. Monteiro and C. M. F. Vieira. “Materials for production of high and ultra-high performance concrete: review and perspective of possible novel materials”. In: *Materials* 15. pp. 4304. 2021.

[3] R. Rajeshwari and S. Mandal. “Prediction of compressive strength of high-volume fly ash concrete using artificial neural network”. In: *Sustainable Construction and Building Materials* 20.4. pp. 471-483. 2019.

[4] A. Farzampour. “Compressive behavior of concrete under environmental effects”. In: *Sustainable Construction and Building Materials* pp. 92-104. 2019.

[5] A. Akhnoukh. “Overview of Concrete Durability Evaluation Using Electrical Resistivity”. In: *Collaboration and Integration in Construction, Engineering, Management and Technology* pp. S. 9-14. 2004.

[6] S. Czarnecki, M. Shariq, M. Nikoo and Ł. Sadowski. “An intelligent model for the prediction of the compressive strength of cementitious composites with ground granulated blast furnace slag based on ultrasonic pulse velocity measurements”. In: *Measurement* 172 pp. 108951. 2021.

[7] S. Propovics. “Analysis of concrete strength versus water-cement ratio relationship”. In: *Materials Journal* 87.5. pp. 517-529. 1990.

[8] M. F. M. Zain, S. M. Abd, K. Sopian, M. Jamil and A. I. Che-Ani. “Mathematical regression model for the prediction of concrete strength”. In: *WSEAS International Conference. Proceedings. Mathematics and Computers in Science and Engineering* 10. 2008.

[9] W. C. Zhu, J. G. Teng and C. A. Tang. “Mesomechanical model for concrete. Part I: model development”. In: *Magazine of concrete research* 56.6, pp. 313-330. 2004.

[10] H. Ling, C. Qian, W. Kang, C. Liang and H. Chen. “Combination of Support Vector Machine and K-Fold cross validation to predict compressive strength of concrete in marine environment”. In: *Construction and Building Materials* 206. pp. 355-363. 2019.

[11] N. D. Hoang, A. D. Pham, Q. L. Nguyen and Q. N. Pham “Estimating compressive strength of high performance concrete with Gaussian process regression model”. In: *Advances in Civil Engineering*. 2861380. 2016.

[12] J. Yu, R. Pan, and Y. Zhao. “High-Dimensional, Small-Sample Product Quality Prediction Method Based on MIC-Stacking Ensemble Learning”. In: *Applied Sciences* 12.1. pp. 23. 2021.

[13] P. Refaeilzadeh, L. Tang and H. Liu. “Cross-validation”. In: *Encyclopedia of database systems* 5. pp. 532-538. 2009.

[14] I-C. Yeh. “Modeling of strength of high performance concrete using artificial neural networks”. In: *Cement and Concrete Research* 28.12. pp. 1797-1808. 1998.

[15] I-C. Yeh. “Modeling slump flow of concrete using second-order regressions and artificial neural networks”. In: *Cement and Concrete Composites* 29.6. pp. 474-480. 2007.

[16] F. Pedregosa et al. “Scikit-learn: Machine learning in Python”. In: *The Journal of machine Learning research* 12. pp. 2825-2830. 2011.

[17] T. Chen and C. Guestrin. “XGBoost: A Scalable Tree Boosting System”. In: *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* pp. 785-794. 2016.

[18] G. Ke et al. “Lightgbm: A highly efficient gradient boosting decision tree”. In: *Advances in neural information processing systems* 30. pp. 3149–3157. 2017.

[19] R. Tibshirani. “Regression shrinkage and selection via the lasso”. In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1. pp. 267-288. 2018.

[20] R. M. Rifkin and R. A. Lippert. “Notes on regularized least squares”. MIT-CSAIL-TR-2007-025. CBCL-268. 2007.

[21] K. P. Murphy “Machine learning: a probabilistic perspective”. MIT press. 2012.

[22] S. J. Kim, K. Koh, M. Lustig, S. Boyd and D. Gorinevsky. “An interior-point method for large-scale ℓ_1-regularized least squares”. In: *IEEE journal of selected topics in signal processing* 1.4. pp. 606-617. 2012.

[23] C. C. Chang “A library for support vector machines”. URL: *ttp://www. csie.ntu.edu.tw/ cjlin/libsvm* 1.4. 2001.

[24] E. Fix and J. L. Hodges. “Discriminatory analysis. Nonparametric discrimination: Consistency properties”. In: *International Statistical Review* 57.3. pp. 238-247. 2001.

[25] C. E. Rasmussen. “Gaussian processes in machine learning”. In: *Summer school on machine learning* pp. 63-71. 2003.

[26] M. Dumont, R. Marée, L. Wehenkel and P. Geurts. “Fast multi-class image annotation with random subwindows and multiple output randomized trees”. In: *International Conference on Computer Vision Theory and Applications (VISAPP)*. 2009.

[27] L. Breiman. “Random forests”. In: *Machine learning* 45.1. pp. 5-32. 2009.

[28] P. Geurts, D. Ernst and L. Wehenkel. “Extremely randomized trees”. In: *Machine learning* 63.1. pp. 3-42. 2006.

[29] Y. Freund and R. E. Schapire. “A decision-theoretic generalization of on-line learning and an application to boosting”. In: *Journal of computer and system sciences* 55.1. pp. 119-139. 1997.

[30] J. H. Friedman. “Greedy function approximation: a gradient boosting machine”. In: *Annals of statistics* pp. 1189-1232. 2001.

[31] A. Guryanov. “Histogram-based algorithm for building gradient boosting ensembles of piecewise linear decision trees”. In: *International Conference on Analysis of Images, Social Networks and Texts* pp. 39-50. 2001.

[32] T. Head et al. “scikit-optimize: v0.5.2. Version v0.5”. URL: *https://pypi.org/project/scikit-optimize/* 2018.

[33] J. Bergstra, R. Bardenet, Y. Bengio and B. Kégl. “Algorithms for hyper-parameter optimization”. In: *Advances in neural information processing systems* 24. 2011.

[34] R. Mustapha and E. A. Mohamed. “High-performance concrete compressive strength prediction based weighted support vector machines”. In: *International Journal of Engineering Research and Applications* 7.1. pp. 68-75. 2017.

[35] D. C. Feng et al. “Machine learning-based compressive strength prediction for concrete: An adaptive boosting approach”. In: *Construction and Building Materials* 230. 117000. 2020.

Spatial Temporal Transformer Networks for Sparse Motion Capture Applications

Lukas Vollenkemper, Martin Kohlhase

Fachhochschule Bielefeld, Center for Applied Data Science
Schulstraße 10, 33330 Gütersloh
E-Mail: {lukas.vollenkemper, martin.kohlhase}@fh-bielefeld.de

1 Introduction

Motion Capture technologies provide the possibility to record human motions. In the past, these systems where primarily used by the movie industry, in order to make realistic computer animations. But in recent years their use in the industrial context gained more and more attention. Recording workers motions provides invaluable information that can be used in many applications. These include the improvement of the ergonomics at the workplaces and more sophisticated human-machine interactions [19]. Motion Capture has been done with numerous types of sensor technologies. Optical systems with RGB or Depth Cameras provide an easy to install method [5]. Optical systems can also use infra-red reflectors on the human body as guidance [1]. Inertial systems use glycerometers, accelerometers and electromagnetic trackers to capture human movement [8]. These systems are more suitable to the industrial context since they do not require a line of sight. On the other hand, inertial systems are expensive and need longer setup times due to the large number of sensors needed to provide accurate recordings. For this reason researchers have developed different solutions to make motion capture recordings from a small number of sensors. This is called sparse motion capture. By reducing the number of sensors needed for accurate measurement, two of the major challenges regarding inertial systems are tackled. It reduces the costs of the system as well as the setup time required in preparation of a recording. Thus, sparse motion capture

is a possibility to foster the usage of motion capture systems in the industrial context. In the past recurrent neural networks as well as transformer networks have been used for sparse motion capture setups. Transformer networks use attention layers to model relationships between input elements. In this work the performance of an alternative attention formulation, known as spatial temporal attention inside of the transformer networks is tested.

2 Sparse Motion Capture

There are approaches to sparse motion capture from different research areas. Some rely on kinematic body models [3]. But many others use machine learning approaches to accomplish sparse motion capture. In [5], an approach to sparse motion capture based on neural networks is presented. A recurrent neural network (RNN) is trained to estimate a pose from only six sensors. Instead of directly determining the position and orientation of individual limbs, parameters of a simplified human body model, the Skinned Multi-Person Linear Model [12] (SMPL) are determined. The authors show that real-time prediction is possible using the approach. It is also shown that a bi-directional RNN architecture produces better results than a classical RNN.
Long-Short-Term Memory Netowrks (LSTM) as a transformer network are used in [4]. The autors furthermore provide a public dataset with motion capture recordings. Data from six sensors serve as input in this approach es well. Here, the sensors sit on the hips, forearms, head, and knees. In the experiments, the LSTM and Transformer architectures are tested against each other with different sensor configurations. Both architectures show good results. The best configuration in their test is sternum, both wrists, lower legs and hip.
In [2] a RNN is trained to generate input parameters for the SMPL body model, similar to [5]. Their approach not only predicts the human pose, but also provides an uncertainty estimation. Their sensor setup consists of six sensors at the wrists, lower legs, head and hip. Another approach to sparse motion capture is presented in [11]. Here, an ensemble of LSTM models is used. Six sensors are used as well, but here the sensors are located at the forearms and at the feet instead of wrists and lower legs. The ensemble consists of bi-directional LSTM

networks, each of which provides an estimate of the pose based on information from one sensor. The individual estimates are then combined to estimate the pose.
In [14], graph convolutional neural networks (GCNN) are tested as an alternative to the RNNs used in [5]. The sparse setup of the sensors is identical and parameters of the SMPL body model are also used as the target. The GCNN allows the topology of the human skeleton to be directly represented in the model. Thus, the authors show that they achieve more accurate pose estimation results than the reference architecture of [5].
In [7], a method is presented that also uses the SMPL model and determines the input parameters based on the electromagnetic signals. They use a recurrent neural network to predict the SMPL Parameters, but the method is limited to electromagnetic sensors.

3 Transformer Networks

Transformer networks and the attention mechanism have become popular since [16] showed that similar or better performance was achieved over recurrent networks in the area of language processing. At the same time, transformer architectures offer the advantage of being parallelizable, which significantly reduces training times. This makes transformer networks an interesting alternative to the recurrent architectures discussed above. The most important building block of the Transformer architecture is the attention mechanism. Via attention, the network learns to understand relationships between individual parts of the sequence. In the language processing domain, this could be the relationship between subject and predicate or verb and adverb. The most widely used version of the attention mechanism is the scaled-dot-product attention described in [16]). The way it works is shown in equations (1-6) The input for each attention layer is a matrix $\mathbf{X} \in \mathbb{R}^{S \times d}$, where S denotes the sequence length and d the length of each sequence element. The layer has three learned parameters $\mathbf{W}_{\text{value}} \in \mathbb{R}^{d \times d_{\text{value}}}$, $\mathbf{W}_{\text{key}} \in \mathbb{R}^{d \times d_{\text{key}}}$ and $\mathbf{W}_{\text{query}} \in \mathbb{R}^{d \times d_{\text{key}}}$ The query, key and value matrices can then be calculated as

$$\mathbf{V} = \mathbf{X}\mathbf{W}_{\text{value}}, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_{\text{key}}, \quad \mathbf{Q} = \mathbf{X}\mathbf{W}_{\text{query}} \tag{1}$$

In order to calculate the attention matrix the key and query matrices are multiplied and scaled with the square root of d_{key}. The scaling is done mainly for numeric stability [16]. This yield the logits matrix $\mathbf{L}$ as described in equation (2). In the later application it is sometimes necessary to prevent certain information to flow through the attention layer. Thus the attention can be masked with a mask $\mathbf{M} \in \mathbb{R}^{S \times S}$ (equation (3)). That is a simple matrix with ones in the lower triangle and negative infinity values in the upper triangle. It is important to note that this is an optional operation that can be skipped.

$$\mathbf{L} = \frac{\mathbf{Q}\mathbf{K}^{\mathrm{T}}}{\sqrt{d_{\text{key}}}} \tag{2}$$

$$\mathbf{L}_{\text{masked}} = \mathbf{L} \odot \mathbf{M} \quad (optional) \tag{3}$$

Afterwards a softmax operation is done over each row of the logits matrix (4, 5). The output of this softmax operation is called attention weights. They are multiplied with $\mathbf{V}$ to get the encodings of the attention layer.

$$\sigma_i(z) := \frac{e^{z_i}}{\sum_{j=1}^{S} e^{z_j}} \tag{4}$$

$$\mathbf{S} = (\sigma(\mathbf{l}_1), \sigma(\mathbf{l}_2), ..., \sigma(\mathbf{l}_S))^{\mathrm{T}} \tag{5}$$

$$\mathbf{O} = \mathbf{S}\mathbf{V}\mathbf{W}_{\text{Out}} \tag{6}$$

The output of the softmax $\mathbf{S}$ multiplied by $\mathbf{V}$ will be of shape $\mathbb{R}^{S \times d_{\text{value}}}$. Another liniear projection $\mathbf{W}_{\text{Out}} \in \mathbb{R}^{d_{\text{value}} \times d}$ is used to receive the final output of the attention layer, which will have the same size as the input sequence $\mathbf{X}$ (equation (6)). In [16], it is proposed to train multiple parallel attention heads rather than performing this process only once. Each head thereby has its own projections for $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ and can thus focus on detecting specific relation types. This is usually called multi-head attention. Standard transformer networks follow an encoder-decoder structure shown in Figure 1.

On the left is the encoder block, into which the input sequences are given. The right block is the decoder, which processes the encoder results together with the previous output variables. In Positional Encoding, information about

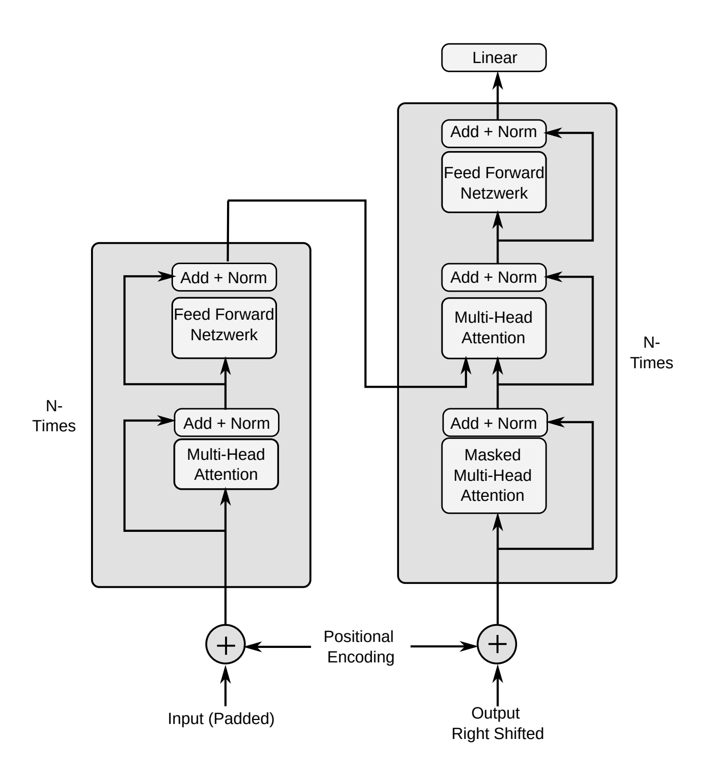


Figure 1: Architecture of the transformer network, [16]

the position of each element in the sequence is added to the network. This is necessary because transformers, unlike recurrent networks, perform the individual prediction steps in parallel during training. The inherent order of the input data is therefore lost. Usually, sine and cosine functions are used as positional encoding [16]. In the encoder, the input sequences are processed by an attention layer and a feed-forward network. This process can be repeated several times before the processed input sequence is passed to the decoder. In the decoder, the previous outputs are processed first. Since the entire sequence is processed in parallel, the values are masked in the attention layer. By masking, only information from previous time steps is used for each time step. This is followed by another attention layer where the encoder results are used as attention. More precisely, the encoder results are used as a inputs

to compute the **K** and **V** matrices, while the results of the previous masked decoder attention are used to compute **Q**. Then, as in the encoder, a feed-forward network is also used. The decoder blocks are also stackable. Skip connections are present throughout the network, where the input of a layer is added to its output. The result is then batch normalized [6]. During training, the encoder block is given the input sequences. The decoder block also receives the sequences, but shifted one position to the right. In the first decoder attention layers, the attention weights are masked such that that only previous values can be used to determine the respective output. Through this masking, the training can be parallelized, which significantly increases the training speed. The encoder creates an encoded version of the input sequence. The decoder then creates the prediction of the next element in the sequence based on this encoding and the previous elements.

4 Spatial Temporal Transformer Networks

In the original transformer architecture it is assumed that each element of the in- and output sequences can be described with a single vector. In NLP these vectors usually come from word embedding, where the single dimensions are not easily interpretable. However in motion capture applications the sequence elements have a inherent structure. They can be grouped by point on the human body, where each point can have a number of measurement values (coordinates, acceleration values, orientation quaternions, etc.). This structure is ignored when the input and output measurements are simply flattened to a vector. *Spatial-Temporal Transformers* (STT) therefor extend the classical transformer architecture by splitting the encoder block into two separate streams that independently consider spatial and temporal aspects of the input data. In the spatial stream, the parameters for each joint are processed in relation to all other joints at the same time. In the temporal stream, the parameters of each joint are analyzed over time, independently of the other joints. STT networks are already used for various tasks. In [9], an STT network for image processing is presented. Other architectures deal with the use in traffic modeling [18] or crime analysis [17]. STT networks are also already being used in human motion analysis, particularly in the area of action recognition.

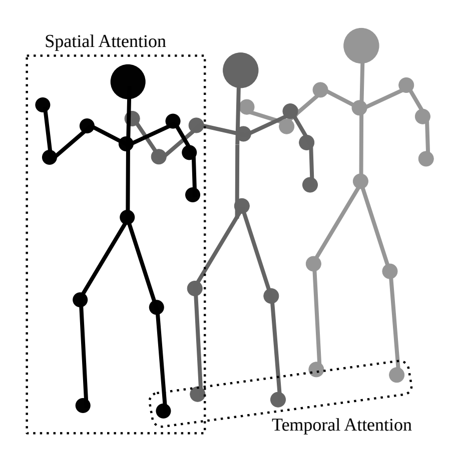


Figure 2: Illustration of the split in temporal and spatial properties

In [20], an approach to action recognition is presented that is particularly robust to noise or missing information. Here, additional tokens are given to each joint at each time step, such as what type of joint it is and the number of the time step.
In [13] Graph Convolutional Networks (GCN) and Convolutional Networks (CNN) from [15] are used to infer such information from the data directly. Furthermore, the authors present the spatial self-attention (spatial encoder) and temporal self attention (temporal encoder) layers. These layers will be tested as an alternative to the classic attention mechanism for sparse motion capture.

Inputs to the spatial encoder are the joint parameters $\mathbf{X} \in \mathbb{R}^{J \times T \times C_{\text{in}}}$. Here T denotes the time steps, J the joints, and C_{in} the number of measurements per joint. The temporal encoder takes $\mathbf{X}^{\text{T}} \in \mathbb{R}^{T \times J \times C_{\text{in}}}$ as inputs. The same formulas as in (1-6) apply to both encoders. However the size of the individual matrices changes slightly. When multiplying three-dimensional tensors with matrices the n-Mode product with $n = 1$ is used [10]. $\mathbf{W}_{\text{key}}$ and $\mathbf{W}_{\text{query}}$ are now in $\mathbb{R}^{C_{\text{in}} \times d_{\text{key}}}$ and $\mathbf{W}_{\text{value}}$ in $\mathbb{R}^{C_{\text{in}} \times d_{\text{value}}}$. The output projection is a matrix $\mathbf{W}_{\text{out}}$ in $\mathbb{R}^{d_{\text{value}} \times C_{\text{in}}}$ in both cases. For the spatial encoder the result of formula (2) will be in $\mathbb{R}^{T \times J \times J}$. That way the network models the importance of different body

joints for each other per time frame. For the temporal encoder however the logits will be in $\mathbb{R}^{J\times T\times T}$. The model can use this to select important time steps for each single time step.
While this split introduces spatial and temporal versions of the projection matrices, the matrices will often be smaller. The number of learned parameters in the classical attention module is $JC_{\text{in}}(2d_{\text{value}}+2d_{\text{key}})$. That is because the C_{in} measurements per joint J have to be described in a single vector in the former version version. Number of learned parameters in the spatio-temporal formulation is $2C_{\text{in}}(2d_{\text{key}}+2d_{\text{value}})$. Thus, the spatio-temporal version uses less parameters as long as more than two joints are analyzed and the key and query dimension is the same. Another useful property is that the number of parameters does not change with the number of joints, as in the spatio-temporal attention formulation. But it also introduces a limitation of this formulation: the same matrices are used over all joints or time steps which might be a limitation to the accuracy.

5 STT for Sparse Motion Capture

After STT networks have been successfully applied in the field of action recognition, it is promising to apply them to pose estimation as well, since they already showed to be able to analyze human motion well. Especially when generating realistic sequences by sparse motion capture, the captured pose must also make sense in combination with the other poses of the sequence. Explicitly modeling the motion of individual joints over time should improve results here. By modeling the spatial relationships in each time segment, the network gets the opportunity to learn and map the structure and anatomy of the human body. To test the applicability of the attention modules presented in [13] they are integrated into the classical Transformer architecture. This is illustrated in Figure 4.

The spatial- and temporal attention layers replace here the classical self attention. Because in this case two layers are used, whose output size corresponds again to the input size, the results must be merged for the skip connection. In this work the outputs of both attention layers are added. In the decoder,

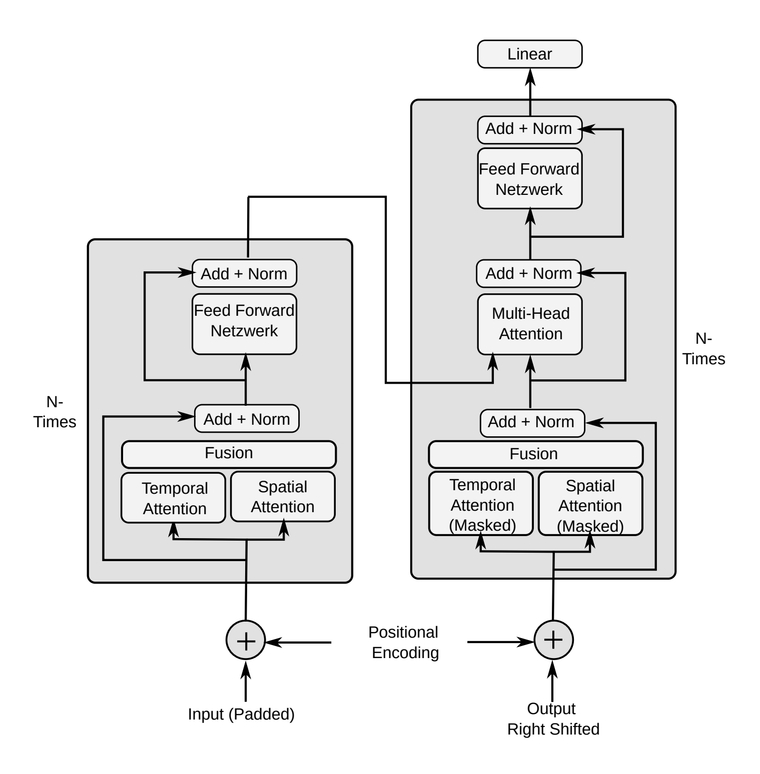


Figure 3: Integration of the spatio temporal encoders into the transformer network structure

temporal attention must be masked so that only the previous values are used for each time point. In the spatial stream the masking is done over the time dimension, such that only the encoding from previous times steps are accessible for the prediction of the current time step. Subsequently, the classical attention mechanism is used to integrate the information from the encoder in the decoder. Since the results from spatial and temporal attention are already combined in the decoder, no further subdivision is done. The rest of the architecture is identical to the classical transformer.

6 Data Sources

Two data sources are used in the experiments. First, Virginia Tech University (VTU) provides a dataset described in [4]. This contains data from 17 participants who were observed using an XSense motion capture suit. Four participants were employees at a hardware store who recorded part of their shift (specifically material handling in the warehouse). The 13 remaining participants were VTU students who recorded their daily routine on campus. A wide variety of activities have been performed. From sitting in the seminar, to walking and standing, to doing sports exercises. This results in a total of approximately 40 hours of motion capture recordings with a resolution of 240 measurements per second.

Secondly, recordings were made with the same system at CfADS Gütersloh. One subject was observed for a total of 3 hours and 47 minutes. This also included a variety of tasks to ensure the greatest possible variance. Among others, these included sports exercises, household activities, working at a desk and eating.

All data has been read from the X-Sense system and reduced to 8 measurements per second. The X-Sense data then contains the position of a total of 64 points on the body. In this work, 27 of these points are used as output variables. These points were selected because they are needed for a later application. The X-Sense system provides position values as well as acceleration or rotation measurements. The approach presented here uses only the 3d coordinates of all 27 points. Thus, it can be easily transferred to other motion capture systems, e.g. optical. In total, 225466 training sequences with 32 elements each are created. In addition, there are 6704 validation sequences that are not used in the training. Six sensor positions with three coordinates each serve as input variables. The output variables are the 27 positions with the corresponding coordinates. Note that this means that the input values are part of the target positions as well. This setup is similar to [4] and could be useful in further research to model noisy input data.

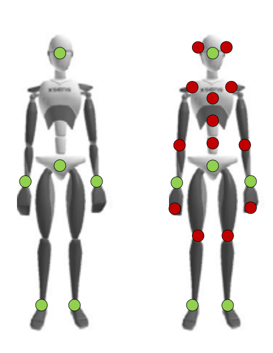


Figure 4: Input of the network (left) and desired outputs (right)

7 Experiments and Benchmark

In this section, the presented architecture will be compared with a classical transformer with respect to performance in sparse motion capture. In [4] it is already shown that the classical transformer architecture can achieve good results in sparse motion capture tasks. The code was used in order to generate predictions from a classical transformer architecture, denoted as VT NMP (Virginia Tech Natural Motion Processing). For this purpose, the two models are first trained five times with different (random) initialization of the parameters. The results shown here are based on the model with the mean validation error. Both models are initialized with three attention heads and the feed forward networks have 128 neurons in the hidden layers. Dropout of 0.01 is used in the attention layers to make the training more robust. The input data is the position of the hips, ankles, wrists and head.

In order to test the model performance a inference procedure slightly different from the forward pass in the training is used. As mentioned above during the training the transformer receives a sequence of target values. In the later application however these values will not be known. Instead the models always predict the next element of the sequence, with their former predictions as target vector. For the first element a padding vector is used as target input. The test is done with all sequences in the validation data set with both a vtnmp model and the spatio-temporal transformer. The mean quadratic error of the estimated

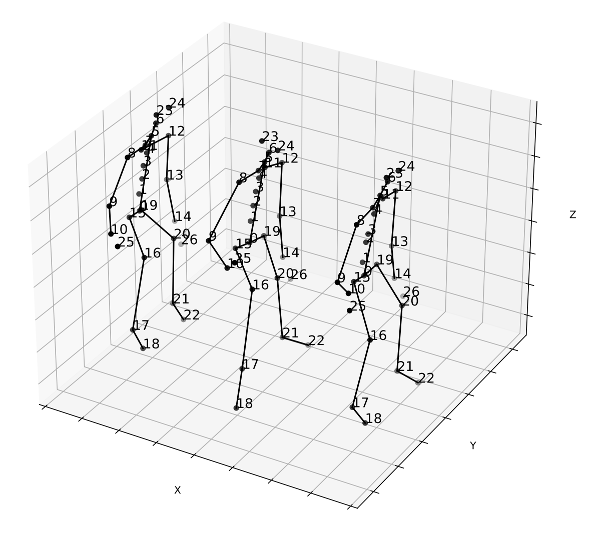


Figure 5: From left to right: VTNMP Prediction, Actual Pose, STT Prediction

point positions per sequence is shown in Figure 6. The error reported can not be interpreted in centimeter, since the batch normalization layers in the networks change the scale. The STT shows a larger error on average as well as a larger number of outliers with large errors.

Figure 7 compares the mean squared error of both networks regarding the single joints over all validation sequences. Points on arms and legs are more difficult to estimate than the spine and hip positions. Hands and toes are the most challenging body parts. STT has problems predicting the position of the right hand. A possible explanation is that the right hand is used in a larger variety of situations. STT also produces large errors at positions, that where originally given to the network (Feet and Hands). This problem in copying of the information of certain joints might be due to the fact that STT uses the same matrices for all joints, which differs from the classical transformer, where each joint, measurement combination has its on parameter in the query, key and value projections.

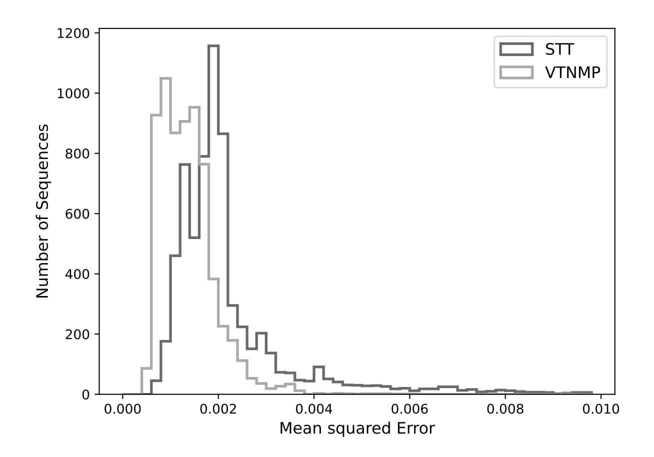


Figure 6: Mean Squared error per Test-Sequence

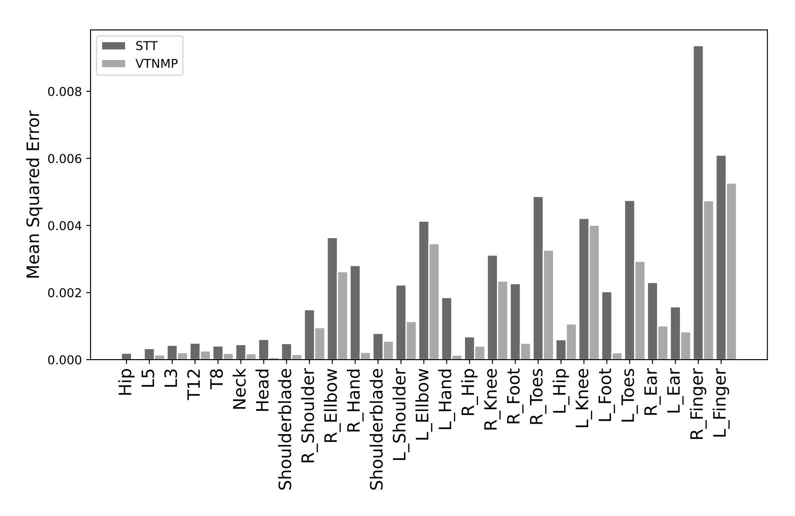


Figure 7: Error per Body Part

8 Conclusion

The aim of this article was to test the applicability of spatio-temporal transformer architectures in sparse pose estimation problems. STT architectures had shown good results in other areas of human motion analysis, especially action recognition. However, it must be concluded that they are not a useful replacement for the multi-head attention layers for sparse motion capture. The performance was tested against an approach using the vanilla attention layers. STT showed that it produces a higher placement error on average. Therefor it must be concluded that analyzing the spatial and temporal information separately might prevent the network from depicting some characteristics of human motion, that are crucial to predict human motion. A possible research direction are more sophisticated versions of the fusion of both streams to enhance the information exchange between temporal and spatial layers. If this hurdle is overcome, the spatial-temporal attention layers provide a useful instrument, because their parameter space does not grow with the number of joints.

References

[1] Accuracy of human motion capture systems for sport applications; state-of-the-art review. *European journal of sport science*, 18(6):806–819, 2018.

[2] Hammad Tanveer Butt, Bertram Taetz, Mathias Musahl, Maria A. Sanchez, Pramod Murthy, and Didier Stricker. Magnetometer robust deep human pose regression with uncertainty prediction using sparse body worn magnetic inertial measurement units. *IEEE Access*, 9:36657–36673, 2021.

[3] Karsten Eckhoff, Manon Kok, Sergio Lucia, and Thomas Seel. Sparse magnetometer-free inertial motion tracking – a condition for observability in double hinge joint systems. *IFAC-PapersOnLine*, 53(2):16023–16030, 2020.

[4] Jack H. Geissinger and Alan T. Asbeck. Motion inference using sparse inertial sensors, self-supervised learning, and a new dataset of unscripted human motion. *Sensors (Basel, Switzerland)*, 20(21), 2020.

[5] Yinghao Huang, Manuel Kaufmann, Emre Aksan, Michael J. Black, Otmar Hilliges, and Gerard Pons-Moll. Deep inertial poser. *ACM Transactions on Graphics*, 37(6):1–15, 2018.

[6] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift, 2015.

[7] Manuel Kaufmann, Yi Zhao, Chengcheng Tang, Lingling Tao, Christopher Twigg, Jie Song, Robert Wang, and Otmar Hilliges. Em-pose: 3d human pose estimation from sparse electromagnetic trackers. pages 11510–11520, 2021.

[8] Sunwook Kim and Maury A. Nussbaum. Performance evaluation of a wearable inertial motion capture system for capturing physical exposures during manual material handling tasks. *Ergonomics*, 56(2):314–326, 2013. PMID: 23231730.

[9] Woojae Kim, Jongyoo Kim, Sewoong Ahn, Jinwoo Kim, and Sanghoon Lee. Deep video quality assessor: From spatio-temporal visual sensitivity to a convolutional neural aggregation network. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[10] Kolda, T. and Bader B. Tensor Decompositions and Applications In SIAM Review Vol. 51 n. 3, pages 455-500, 2009.

[11] Deepak Nagaraj, Erik Schake, Patrick Leiner, and Dirk Werth. An rnn-ensemble approach for real time human pose estimation from sparse imus. In Nicolai Petkov, editor, *Proceedings of the 3rd International Conference on Applications of Intelligent Systems*, ACM Digital Library, pages 1–6, New York,NY,United States, 2020. Association for Computing Machinery.

[12] Georgios Pavlakos, Luyang Zhu, Xiaowei Zhou, and Kostas Daniilidis. Learning to estimate 3d human pose and shape from a single color image.

[13] Chiara Plizzari, Marco Cannici, and Matteo Matteucci. Spatial temporal transformer network for skeleton-based action recognition. In *Pattern Recognition. ICPR International Workshops and Challenges*, pages 694–701. Springer International Publishing, 2021.

[14] Patrik Puchert and Timo Ropinski. Human pose estimation from sparse inertial measurements through recurrent graph convolution.

[15] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Two-stream adaptive graph convolutional networks for skeleton-based action recognition. *undefined*, 2019.

[16] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[17] Xian Wu, Chao Huang, Chuxu Zhang, and Nitesh V. Chawla. Hierarchically structured transformer networks for fine-grained spatial event forecasting. In Yennun Huang, Irwin King, Tie-Yan Liu, and Maarten van Steen, editors, *WWW '20*, pages 2320–2330, New York, op. 2020. ACM =Association for Computing Machinery.

[18] Mingxing Xu, Wenrui Dai, Chunmiao Liu, Xing Gao, Weiyao Lin, Guo-Jun Qi, and Hongkai Xiong. Spatial-temporal transformer networks for traffic flow forecasting.

[19] Zuhair Zafar, Rahul Venugopal, and Karsten Berns. Real-time recognition of human postures for human-robot interaction. 2018.

[20] Yuhan Zhang, Bo Wu, Wen Li, Lixin Duan, and Chuang Gan. Stst: Spatial-temporal specialized transformer for skeleton-based action recognition. In Heng Tao Shen, editor, *Proceedings of the 29th ACM International Conference on Multimedia*, ACM Digital Library, pages 3229–3237, New York,NY,United States, 2021. Association for Computing Machinery.

Evaluation of Transformer Architectures for Electrical Load Time-Series Forecasting

Matthias Hertel*, Simon Ott*, Benjamin Schäfer, Ralf Mikut, Veit Hagenmeyer, Oliver Neumann

Institute for Automation and Applied Informatics,
Karlsruhe Institute of Technology
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen
E-mail: matthias.hertel@kit.edu, simon.ott@student.kit.edu
* equal contribution

Abstract

Accurate forecasts of the electrical load are needed to stabilize the electrical grid and maximize the use of renewable energies. Many good forecasting methods exist, including neural networks, and we compare them to the recently developed Transformers, which are the state-of-the-art machine learning technique for many sequence-related tasks. We apply different types of Transformers, namely the Time-Series Transformer, the Convolutional Self-Attention Transformer and the Informer, to electrical load data from Baden-Württemberg. Our results show that the Transformes give up to 11% better forecasts than multi-layer perceptrons for long prediction horizons. Furthermore, we analyze the Transformers' attention scores to get insights into the model.

1 Introduction

Transmission system operators (TSOs) must balance the electricity supply and electrical load in the grid at every moment [1]. Otherwise, the grid becomes

instable, which can lead to electricity outages. In order to plan the dispatch of energy storages and remaining fossil power plants, as well as the control of flexible consumers, accurate forecasts of the electrical load for the next hours to days are needed. Renewable power plants can be curtailed more easily than fossil power plants, which need more time to reduce their generation. Therefore, when the electrical load is overestimated, usually the renewable energy sources get curtailed. This means good forecasts of the electrical load are also necessary to maximize the usage of renewable energy.

Recent work on time-series forecasting showed good results with Transformers [2] for different applications [3, 4, 5, 6], including electrical load forecasting [7, 8, 9, 10, 11, 12]. Transformers can process long sequences and model long-term dependencies with the attention mechanism [2]. Therefore, they have the potential to give good results in electrical load forecasting and work especially well for long prediction horizons. In this work, we analyze whether the Transformer beats multiple baselines in forecasting the electrical load for the German state Baden-Württemberg, and discuss possible future usages of Transformers in energy forecasting.

Our contributions are the following:

- We compare multiple types of Transformers, namely the Time-Series Transformer [3], the Convolutional Self-Attention Transformer [5] and the Informer [6] on forecasting the electrical load of the state Baden-Württemberg.
- We compare the Transformers with multiple baselines, including a load profile baseline, linear regression models and multi-layer perceptrons.
- We analyze the attention scores of one of the Transformer models to get insights into the model's predictions, and we propose Transformer architectures that we want to test in the future based on our experience.
- We make all the code for our experiments publicly available.[1]

[1] github.com/KIT-IAI/Transformer-Networks-for-Electrical-Load-Time-Series-Forecasting

The paper is organized as follows: First, we discuss the related work on electrical load forecasting and Transformers in Section 2. Then, we define the electrical load forecasting task in Section 3. The different Transformer architectures are introduced in Section 4. The experimental setup, results and analysis of the attention scores are described in Section 5. Finally, we conclude in Section 6 with an outlook to future work on the topic.

2 Related Work

Modeling the patterns that underlie social behavior as energy load is difficult, which is why data-driven solutions are used in practice. Classical approaches rely on statistical methods with manually engineered features, such as linear regression, ARIMA and Support Vector Machines. To overcome the manual feature engineering, new methods based on deep learning were developed. González Ordiano et al. [13] give an overview on existing energy time-series forecasting methods, including linear regression and multi-layer perceptrons, which we are going to use as baseline models (see Section 5). More sophisticated methods were developed in the meantime, such as profile neural networks [14].

Transformers [2] were originally developed in the field of Natural Language Processing, where they became the state of the art in many tasks. Transformers use attention to retrieve information from the input time series and are thereby capable of modeling long-term dependencies. Multiple publications adapt the Transformer architecture to overcome specific disadvantages. The Convolutional Self-Attention Transformer [5] combines the attention mechanism with convolutions, to be able to better recognize patterns in the time series. The Informer [6] introduces ProbSparse attention to reduce the time and space complexity of the attention mechanism, and adds convolutions and max-pooling layers which reduce the length of the time series after each encoder layer. Zeng et al. [15] on the other hand question whether Transformers are really effective for time series forecasting. Our goal is to apply the different proposed Transformer architectures to state-level aggregated electrical load data from Baden-Württemberg and compare them against strong baselines.

3 Task Definition

We address the following electrical load forecasting problem: At a time step t, given the hourly electrical load of the previous p time steps $x_{(t-p+1):t} = (x_{t-p+1}, ..., x_t)$, m covariate sequences $z^j_{(t-p+1):t}$ with $1 \leq j \leq m$, and n a priori known covariate sequences $z^l_{(t+1):(t+\tau)}$ with $1 \leq l \leq n$, the goal is to predict the next τ electrical load values $x_{(t+1):(t+\tau)}$. We use one week's values as input (i.e. $p = 168$), and a forecasting horizon of $\tau = 96$ hours. We use time and calendar features as covariates, as explained in detail in Section 5. In the future, the covariates can be extended to cover external data such as weather data.

4 Approach

We use three different Transformer architectures. First, the Time-Series Transformer; second, the Convolutional Self-Attention Transformer; and third, the Informer. The architecture of the Time-Series Transformer is described in Section 4.1. It is the base of the other two Transformer architectures. The differences between the Convolutional Self-Attention Transformer and Informer and the Time-Series Transformer are described in Sections 4.2 and 4.3 respectively. The hyperparameters of the Transformer models are given in Section 5.3.

4.1 Time-Series Transformer

An overview of the Time-Series Transformer architecture is shown in Figure 1. The model consists of an encoder (shown in the left-hand part of the figure) and a decoder (shown in the right-hand part of the figure), both described in the following.

The input to the encoder is a sequence of p vectors, one for each past time step used by the model. Each vector contains one entry for the electrical load and additional entries for the time and calendar features for this time step. Before giving the vectors as an input to the encoder, we run them through a linear layer with d_{model} units, so that the input to the first encoder layer has shape $p \times d_{\text{model}}$,

where d_{model} is the hidden dimension of the Transformer. Each encoder layer attends to the p outputs of the previous layer with the multi-head self-attention mechanism.

The input to the decoder consists of the vectors for the previous p_d time steps and the next τ time steps. The electrical load for the next τ time steps is unknown and therefore set to zero. The vectors are also run through a linear layer to increase the vector size to d_{model}. Each decoder layer attends to the outputs of the previous layer with the multi-head self-attention mechanism. Masking prevents the self-attention from attending to vectors that correspond to future time steps.[2] In addition, each decoder layer attends to the outputs of the last encoder layer with the multi-head cross-attention mechanism. The last τ outputs of the decoder, which correspond to the τ next time steps, are fed into a linear layer with a single unit, resulting in the τ predictions.

A sinusoidal positional encoding is added to the input of the first encoder and decoder layer. This is used in the Transformer [2] to make use of distances and absolute positions in the time series. Since we give time information also as covariates, we learn a weight for the positional encoding and a weight for the input vectors, which are both initialized as one.

4.2 Convolutional Self-Attention Transformer

The Convolutional Self-Attention Transformer differs from the Time-Series Transformer in that it uses convolutional self-attention [5] instead of the normal self-attention [2]. Before computing the keys and queries for the self-attention heads, causal 1D convolution with stride one and kernel size k is applied to the sequence of vectors.

Li et al. [5] additionally propose LogSparse attention to reduce the time and space complexity of the attention. Since we do not notice memory issues in our experiments, we do not make use of the LogSparse attention.

[2] It would be fine to attend to future time steps, since all features are already known at prediction time. However, we kept the masking to be consistent with the standard encoder-decoder architecture.

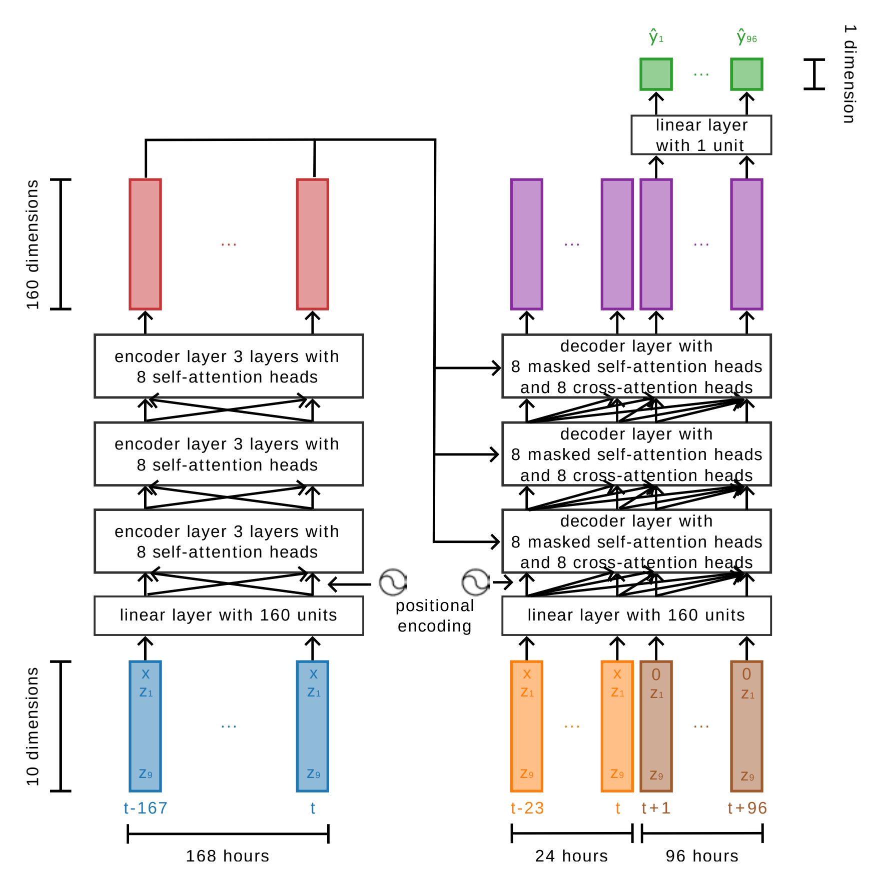


Figure 1: Data flow in the Time-Series Transformer. The architecture consists of an encoder part (left-hand side) and a decoder part (right-hand side). The input vectors to the encoder are shown in blue, and the output of the encoder in red. The decoder receives vectors for the previous day (orange) and next four days (brown). Each decoder layer attends to the encoder output (red) with multi-head cross-attention. Additionally, each encoder and decoder layer attends to its inputs with multi-head self-attention. The decoder output (purple) corresponding to the next day is fed through a linear layer to compute the predicted electrical load (green).

4.3 Informer

Compared to the Time-Series Transformer, the Informer [6] has two additional layers after each encoder layer. The first additional layer is a convolutional

layer. The second additional layer is a max-pooling layer. The additional max-pooling layers cut the length of the time series in half after each decoder layer.

Zhou et al. [6] additionally propose ProbSparse attention to reduce the time and space complexity of the attention mechanism. Since we do not notice memory problems in our experiments, and the results of the Informer were slightly worse with ProbSparse attention, we use normal attention instead.

5 Experiments

Next, we describe the dataset in Section 5.1, the baselines in Section 5.2, the models in Section 5.3, the evaluation metric in Section 5.4, and the results in Section 3.3. Limitations are discussed in Section 5.6. Finally, an analysis of the Time-Series Transformer's attention scores is presented in Section 5.7.

5.1 Dataset

The selected dataset for the experiments is the electrical load of Baden-Württemberg from the Open Power System Data time-series dataset [16].[3] It contains the electrical load in MW for every quarter of an hour from 2015 to 2019. In order to shorten the time series, we transform the data to the hourly resolution by averaging every consecutive four values. We use the data from 2015 to 2017 as the training set, 2018 as the validation set, and 2019 as the test set. This gives 26,016 examples for training, and 8,496 each for validation and test. The dataset is standardized using the mean and standard deviation from the training set.

We use the following time and calendar features as covariates: the hour of the day, the week of the year (both sine- and cosine-encoded), whether the day is a workday, whether the day is a holiday, whether the previous day is a workday, whether the next day is a workday, and whether the day is in the

[3] https://data.open-power-system-data.org/time_series/

Christmas period from December 24th to 27th (all binary). Overall, this makes nine covariates.

5.2 Baselines

We compare the different Transformer architectures with three baselines: a load profile baseline, a linear regression model, and multi-layer perceptrons.

Load profile baseline We create daily profiles by computing the average of the load for every hour of each combination of month and day of the week. This makes twelve times seven daily profiles, each consisting of 24 averaged hourly load values. Holidays are treated like Sundays and seven special daily profiles computed for the two weeks after Christmas. At inference time, the profile corresponding to the day in question is used as predictions.

Linear regression The second baseline is a multi-output linear regression model. It gets the last 168 values of the electrical load time series as input, together with the nine time and calendar features for the first hour to predict, and predicts the next 96 electrical load values. The model has $\tau \cdot (\textit{number of inputs} + 1)$ parameters, which in our case is $96 \cdot 178 = 17,088$.

Multi-layer perceptrons The multi-layer perceptrons (MLPs) use the same inputs as the linear regression models. The MLPs consist of multiple hidden layers with ReLU activation, and an output layer with linear activation with 96 units for the 96 predicted values. Results for MLPs with one to three layers and 256 to 2048 units are reported in Table 1. We choose a small MLP with two layers and 256 units per layer and a large MLP with two layers and 2048 units per layer as baselines for the Transformer models.

Table 1: MAPE on the test set for multi-layer perceptrons with varying numbers of layers and units. The results are averaged across ten runs with different random seeds.

Layers	Units per layer	MAPE [%]
1	256	3.33
1	512	3.22
1	1024	3.17
1	2048	3.15
2	256	3.33
2	512	3.20
2	1024	3.15
2	2048	3.12
3	256	3.34
3	512	3.22
3	1024	3.16
3	2048	3.12

5.3 Models

The Transformer models are the Time-Series Transformer, the Convolutional Self-Attention Transformer and the Informer, described in Section 4. We use vectors for the previous $p = 168$ time steps (i.e. one week) as input to the encoder, and vectors for the previous $p_d = 24$ time steps together with the $\tau = 96$ next time steps as input to the decoder. Each model consists of three encoder and three decoder layers with eight heads for the attention modules. The model dimension d_{model} is set to 160. The kernel size k for the Convolutional Self-Attention Transformer is set to twelve, and the kernel size for the Informer to three. We also tested Transformer models with a different number of layers, and varying model dimensions d_{model} and kernel sizes k, and found this architecture to be optimal among all tested variants.

An overview of the model sizes is given in Table 2. Notably, the large MLP has more trainable parameters than the Transformer models. The Convolutional Self-Attention Transformer and Informer have more trainable parameters than the Time-Series Transformer due to the additional convolutional layers.

Table 2: Model sizes.

Model	Layers	Units or d_{model}	#parameters
Linear regression	-	-	17,088
MLP small	2	256	233,056
MLP large	2	2,048	5,533,792
Time-Series Transformer	3 + 3	160	1,245,605
Conv. Self-Att. Transformer	3 + 3	160	4,013,285
Informer	3 + 3	160	1,400,165

All models are trained with the mean absolute error (MAE) as the loss function using the AdamW [17] optimizer. The initial learning rate is 0.0005, which is decayed by 90% after every two epochs. The batch size is set to 32. Early stopping is used to achieve the lowest possible generalization error on the validation data, with a patience of five epochs. The model with the lowest validation error is saved and used in the evaluation. We find that due to early stopping, the MLPs improve the validation error for no longer than 18 epochs, and the Transformer models for no longer than eight epochs. A possible explanation for the short training is the high number of trainable parameters in the models compared to the few training examples, which lets the models overfit easily.

5.4 Metric

To evaluate the performance of a model, we compute its mean absolute percentage error (MAPE) for forecasts from 1 to τ hours into the future. For each hour t in the test dataset, we have a vector $\hat{y}_t \in \mathbb{R}^\tau$ with the predicted electrical load for the next τ hours, and a vector $y_t \in \mathbb{R}^\tau$ with the actual electrical load of the next τ hours. We denote the ith entry in y_t as $y_{t,i}$ and the ith entry in $\hat{y}_t$ as $\hat{y}_{t,i}$. We evaluate the MAPE for forecasting T hours into the future, called MAPE_T, with $1 \leq T \leq \tau$. It is computed as follows:

$$\text{MAPE}_T(y, \hat{y}) = \frac{1}{N} \cdot \sum_{t=1}^{N} \left| \frac{y_{t,T} - \hat{y}_{t,T}}{y_{t,T}} \right|,$$

where N is the number of examples in the test set.

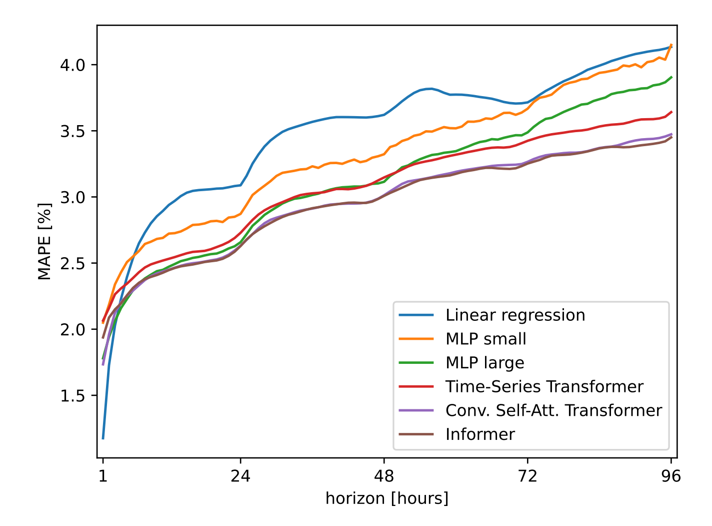


Figure 2: Results of the different models and baselines for predicting the electrical load from one to 96 hours into the future, evaluated on the test set.

5.5 Results

The results of the different models evaluated on the test set for forecasting horizons from one to 96 hours are shown in Figure 2. The results are averaged across ten runs with different random initializations of the neural networks' parameters.

All models are better than the load profile baseline, which has a constant MAPE value of 4.92% (not shown in the figure). For short prediction horizons of one to three hours, the linear regression is the best model. However, its MAPE value increases rapidly and after seven hours it is already the worst model. The large MLP is always better than the small MLP and is the best model for prediction horizons of four and five hours. For prediction horizons of six hours and more, either the Convolutional Self-Attention Transformer or the Informer is the best model. The Time-Series Transformer is worse than the other two Transformer variants, but it is also better than the MLPs for prediction horizons

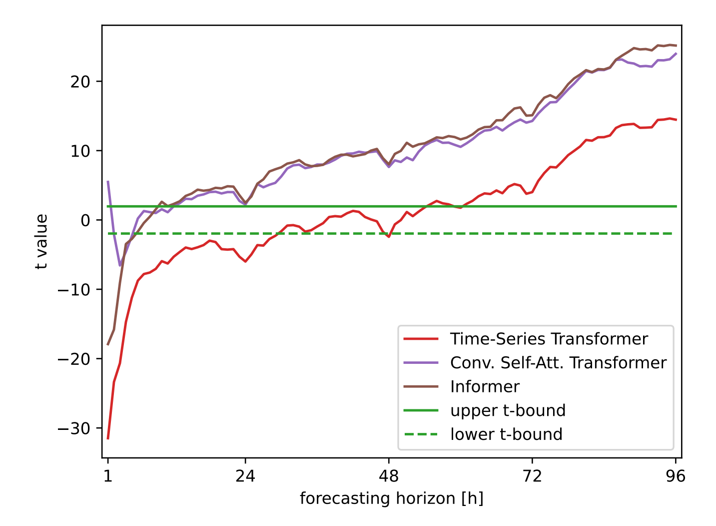


Figure 3: Results of Welch's t-tests with $\alpha = 0.025$ for each forecasting horizon. The Time-Series Transformer, Convolutional Self-Attention Transformer and Informer are compared to the MLP with two layers and 2048 units per layer.

longer than two days. Doing a Welch's t-test with $\alpha = 0.025$, we find that the Informer and the Convolutional Self-Attention Transformer are significantly better than the large MLP after 10 and 12 hours respectively, and the Time-Series Transformer is significantly better than the large MLP after 61 hours (see Figure 3). The Convolutional Self-Attention Transformer and the Informer are significantly better than the Time-Series Transformer for all forecasting horizons.

5.6 Discussion

In our experiments, Transformers beat the baselines for long prediction horizons, which shows their potential in electrical load forecasting. However, a comparison to other machine learning methods, such as random forests, support vector regression, long-short-term memories, convolutional neural networks and profile neural networks [14], must be made in the future. Also, some

of the methods could benefit from feature engineering, feature selection and inclusion of external features such as weather data more than others, which could change the results. We notice that our models have many trainable parameters compared to the small number of training examples, and train only for a few epochs because of early stopping. Other training hyperparameters and more training data could lead to better results. We have compared three Transformer architectures in our work, and would like to include more in the future, for example Temporal Fusion Transformer [4], Autoformer [11] and FEDformer [18]. We have only used one dataset in our work, which contains the electrical load of the state Baden-Württemberg. An evaluation on more datasets, for example on the less aggregated and therefore more volatile load of buildings, and on other forecasting tasks, such as renewable energy generation forecasting, would help to analyze the usefulness of Transformers for energy time-series forecasting.

5.7 Attention Scores

Figure 4 shows an exemplary plot of the input and output time series and the attention scores of the Time-Series Transformer. The last observed hours before the prediction get a high attention when the model predicts the next few hours (see number 1 in the figure). The previous day is attended mostly when the model predicts the next day (2). A diagonal pattern can be seen, which means the model attends the embeddings from about 24 hours before the prediction. The valleys in the time series are attended when the model predicts the electrical load at night (3). Peaks in the time series are attended when the model predicts the electrical load at daytime (4). The patterns for valleys (3) and peaks (4) are similar, but shifted along the y-axis (that is, shifted with the prediction horizon). Similar patterns are seen for the other weekdays of the blue curve. The lowest values at Sunday mornings are always attended, even when the model does not make a prediction for a weekend (5).

We can use the attention scores as a plausibility check for the model. It is reasonable that the last observed hours are important to predict the next few hours, since they will often have similar values. The attention on peaks and valleys can be understood, because the rest of the time series can be inferred

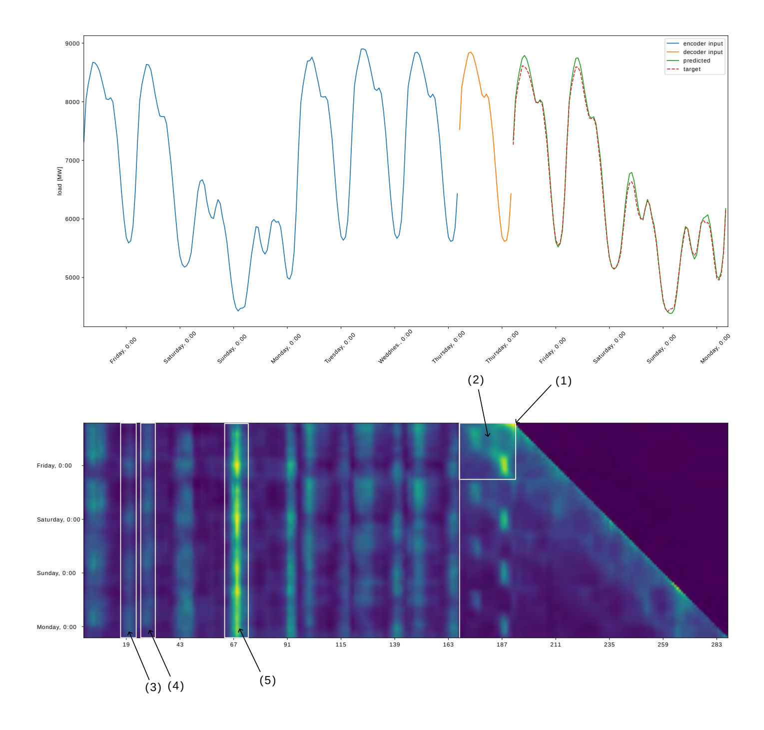


Figure 4: Visualization of the decoder's attention scores averaged across all Thursdays at 5 a.m. that appear in the test set. The upper part shows four averaged time series: the input to the encoder (blue), the previous day fed to the decoder (orange), the predictions (green) and expected values (red). The lower part shows the cross-attention on the left and self-attention on the right. Each row corresponds to one prediction time step, from top to bottom in chronological order. The lighter the color, the higher the attention score. The upper right triangle of the self-attention consists of zeros because of the masked self-attention, that prevents the model from attending future time steps.

from its highest and lowest values. Peaks are important at daytime when the predicted values are high, and valleys at night when the predicted values are low. However, we had expected that the model would attend the previous weekend only while making a forecast for the weekend. In addition, we had

expected more of a diagonal pattern in the cross-attention scores, meaning that the model would attend the same weekday a week ago.

In the future, we want to apply Transformers with multiple time series as input, such as additional weather data, and use the attention scores to estimate feature importance. Another possible direction of future research is to adapt the attention scores such that the Transformer attends more on the previous weekday and less on the previous Sunday, and investigate whether this improves or deteriorates the performance. A third option is to select the features that received high attention, such as the peaks and valleys, and use them in another model. The resulting model could achieve the best of two worlds: the good performance of the Transformer, and the fast training and inference of simpler methods.

6 Conclusion and Future Work

Our experiments showed that Transformers give better electrical load forecasts for the state Baden-Württemberg than multiple statistical baselines and multilayer perceptrons. In the future, a comparison to other strong machine learning methods must be made. We plan to integrate the Transformers into the Python Workflow Automation Tool for Time-Series (pyWATTS) [19] and evaluate them against the models already included in the package. In addition, we want to incorporate external features such as weather data, and evaluate if the model ranking remains the same. Transformers could also be useful for other energy forecasting tasks, such as forecasting the more volatile electrical loads of individual buildings or forecasting renewable energy generation.

The exact details of the Transformer architecture are important to get the best results, as in our experiments the Convolutional Self-Attention Transformer and the Informer were better than the Time-Series Transformer. More architectures from the literature [4, 11, 18] could be added to the comparison, and new architectures developed.

Promising research directions are to use the Transformer's attention scores to better understand the models and get closer towards explainable AI methods, or to use the gained insights to develop new models.

Acknowledgement

This project is funded by the Helmholtz Association's Initiative and Networking Fund through Helmholtz AI.

References

[1] Jan Machowski et al. "Power system dynamics and stability". John Wiley & Sons, 1997.

[2] Ashish Vaswani et al. "Attention is all you need". In: *Advances in Neural Information Processing Systems* 30 (2017).

[3] Neo Wu et al. "Deep transformer models for time series forecasting: The influenza prevalence case". In: *arXiv preprint arXiv:2001.08317* (2020).

[4] Bryan Lim et al. "Temporal Fusion Transformers for Interpretable Multi-horizon Time Series Forecasting". In: *CoRR abs/1912.09363* (2019).

[5] Shiyang Li et al. "Enhancing the locality and breaking the memory bottleneck of transformer on time series forecasting". In: *Advances in Neural Information Processing Systems* 32 (2019).

[6] Haoyi Zhou et al. "Informer: Beyond efficient transformer for long sequence time-series forecasting". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 12. 2021, pp. 11106–11115

[7] Guangqi Zhang et al. "Short-Term Electrical Load Forecasting Based on Time Augmented Transformer". In: *International Journal of Computational Intelligence Systems* 15.1 (2022), pp. 1–11.

[8] Shichao Huang et al. “Short-Term Load Forecasting Based on the CEEMDAN-Sample Entropy-BPNN-Transformer”. In: *Energies* 15.10 (2022), p. 3659.

[9] Alexandra L’Heureux, Katarina Grolinger, and Miriam AM Capretz. “Transformer-Based Model for Electrical Load Forecasting”. In: *Energies* 15.14 (2022), p. 4993.

[10] Chen Wang et al. “A Transformer-Based Method of Multienergy Load Forecasting in Integrated Energy System”. In: *IEEE Transactions on Smart Grid* 13.4 (2022), pp. 2703–2714.

[11] Haixu Wu et al. “Autoformer: Decomposition Transformers with Auto-Correlation for Long-Term Series Forecasting”. In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc’Aurelio Ranzato et al. 2021, pp. 22419–22430.

[12] Zezheng Zhao et al. “Short-Term Load Forecasting Based on the Transformer Model”. In: *Inf.* 12.12 (2021), p. 516

[13] Jorge Ángel González Ordiano et al. “Energy forecasting tools and services”. In: *WIREs Data Mining Knowl. Discov.* 8.2 (2018).

[14] Benedikt Heidrich et al. “Forecasting energy time series with profile neural networks”. In: *Proceedings of the Eleventh ACM International Conference on Future Energy Systems*. 2020, pp. 220–230.

[15] Ailing Zeng et al. “Are Transformers Effective for Time Series Forecasting?” In: *CoRR abs/2205.13504* (2022).

[16] Frauke Wiese et al. “Open Power System Data–Frictionless data for electricity system modelling”. In: *Applied Energy* 236 (2019), pp. 401–409.

[17] Ilya Loshchilov and Frank Hutter. “Decoupled Weight Decay Regularization”. In: *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*.

[18] Tian Zhou et al. "FEDformer: Frequency Enhanced Decomposed Transformer for Long-term Series Forecasting". In: *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*. Ed. by Kamalika Chaudhuri et al. Vol. 162. Proceedings of Machine Learning Research. PMLR, 2022, pp. 27268–27286.

[19] Benedikt Heidrich et al. "PyWATTS: Python workflow automation tool for time series". In: *arXiv preprint arXiv:2106.10157* (2021).

Weißes Rauschen basierte Verlustfunktion zur verbesserten Zeitreihenprognose mit künstlichen neuronalen Netzen

Tobias Fischer, Fabian Bauer, Silas Selzer, Peter Bretschneider

Technische Universität Ilmenau, Institut für Elektrische Energie- und Steuerungstechnik, Fachgebiet Energieeinsatzoptimierung
Gustav-Kirchhoff-Str. 5, 98693 Ilmenau
E-Mail: {tobias-merlin.fischer, fabian.bauer, silas-aaron.selzer, peter.bretschneider}@tu-ilmenau.de

1 Einleitung

Zeitreihenprognosen stellen ein wichtiges Werkzeug für eine sichere und stabile Energieversorgung dar. Exakte Vorhersagen von Energielastgängen bilden die Grundlage für eine optimale Planung und Steuerung des elektrischen Energiesystems. Zur möglichst genauen Vorhersage ist es notwendig, Modelle zu entwickeln und zu untersuchen, die dem wachsenden Mengengerüst an messtechnisch erfassten Energiedaten und exogenen Einflussgrößen Rechnung tragen. Vor diesem Hintergrund spielen Verfahren des maschinellen Lernens eine wichtige Rolle bei der Modellierung linearer und nicht-linearer Zeitreihen. Die Qualität und Quantität der Informationsgewinnung wird beim maschinellen Lernen über die verwendete Verlustfunktion bewertet. In den durchgeführten Untersuchungen werden drei überarbeiteteVerlustfunktionen basierend auf weißem Rauschen entwickelt und mit bisherigen Methoden verglichen.[1, 2, 3]

2 Weißes Rauschen und Periodogramm

Der Einsatz des Gradientenabstiegsverfahrens im maschinellen Lernen garantiert nur die Konvergenz zu einem stationären Punkt, was sich als problematisch erweisen kann. Typische Verlustfunktionen, wie bspw. der mittlere quadratische Fehler (MSE), stellen ein Optimalitätskriterium basierend auf dem Amplitudenabstand der Vorhersage und den Originaldaten dar. Dabei ist zu bedenken, dass es Ziel der Regressionsanalyse ist, Muster in einer Zeitreihe zu erlernen, der MSE aber nur deren Ausprägung als Zielwert definiert. Um die Zeitachse als Zieldimension mitzubetrachten, wird der MSE um die Untersuchung des Residuums, also der Differenz aus Prognose und Originaldaten, auf weißes Rauschen erweitert. Eine Vorhersage kann als optimal angenommen werden, wenn das Residuum weißem Rauschen entspricht und somit das Restsignal keine Informationen enthält. In dieser Arbeit wird weißes Rauschen mit Gauß'schen weißen Rauschen gleichgesetzt. Gaußsches weißes Rauschen ist ein Prozess, dessen Zufallsvariablen ε_t normalverteilt sind und alle den gleichen Erwartungswert $\mu = E(\varepsilon_t)$ und Varianz $\sigma^2 = Var(\varepsilon_t)$ haben. Er ist der einfachste schwachstationäre Prozess, Mittelwert- und Varianzfunktion sind hier für alle t konstant $\mu(t) = \mu$, $\sigma^2(t) = \sigma$. Der Vorteil des weißen Rauschens gegenüber anderem Rauschen, bspw. dem $1/f$-Rauschen ist die stochastische Unabhängigkeit der Zufallsvariablen. Damit gilt, für $s \neq t$, $E(\varepsilon_s \varepsilon_t) = E(\varepsilon_s)E(\varepsilon_t)$, voraus $Cov(\varepsilon_s, \varepsilon_t) = 0$ folgt. Das Spektrum des weißen Rauschens ist konstant, alle Frequenzen liefern den gleichen Beitrag zur Prozessvarianz. Dieses konstante Spektrum ermöglicht es mittels geeigneter Leistungsdichteschätzer einen Test auf weißes Rauschen durchzuführen. Eine Teststatistik zur Überprüfung der Hypothese, ob der untersuchte Prozess weißem Rauschen entspricht, ist das sogenannte kumulierte Periodogramm. Für den genauen Ablauf der Teststatistik wird auf Unterabschnitt 3.2 verwiesen. [4, 5, 6]

3 Untersuchungsansätze

Die Prognose der Energielastgänge erfolgt mittels künstlicher neuronaler Netze und als Netzarchitektur wird ein Multi-Layer-Perceptrons (MLP) verwendet.

Die Vorhersage der gesuchten Größe zum nächsten Zeitschritt $Y(t_n)$ erfolgt aus den letzten N Werten der Eingangsvariablen $(X(t_{n-1})\ldots X(t_{n-N}))$. Mit $X \subseteq \mathbb{R}^{d_x}$ als Eingaberaum und $Y \subseteq \mathbb{R}^{d_y}$ als Ausgaberaum ergibt sich als Ziel des überwachten Lernens eine zu minimierende Verlustfunktion $L((x;\theta);y)$, wobei θ sowohl die Parameter (Gewichte), als auch Hyperparameter bezeichnet.

3.1 Mean Squared Error

Eine der am häufigsten eingesetzten Verlustfunktionen in der Regression ist der Mean Squared Error, aufgrund der erhöhten Bestrafung von stärkeren Abweichungen und wird berechnet durch [7]:

$$MSE = \frac{1}{N}\sum_{n=1}^{N}\left(x_{n_{\text{pred}}} - y_{n_{\text{true}}}\right)^2 \tag{1}$$

3.2 Weißes Rauschen basierte Verlustfunktionen

Der hier verwendete Ansatz erweitert den MSE um einen Summanden, der den Fehler zwischen Vorhersage und Realwerten nicht nur als Distanzdifferenz berechnet, sondern das Residuum $R_n = x_{n_{pred}} - y_{n_{pred}}$ mit $n = 1,\ldots N$ auf weißes Rauschen untersucht. Der Einflussfaktor α entspricht dem relativen Einfluss der erweiterten Verlustfunktion auf den MSE. β ist ein Skalierungsfaktor, der Größenunterschiede zwischen beiden Funktionen ausgleicht. Es werden drei Möglichkeiten untersucht, die Gleichheit des Residuums und weißem Rauschen für das KNN zu bestimmen. Indem ein Signal W als ein Signal Gauß'schen weißen Rauschens definiert wird, mit Standardabweichung und Mittelwert identisch zu R, kann mit der Fourierfrequenz $\lambda_n = n/N$ und Autokovarianzfunktion c_τ für $r = 1,\ldots,N$ die Verlustfunktion mittels Fourier Transformation erweitert werden. [5, 8]

3.3 Kumuliertes Periodogramm

Das kumulierte Periodogramm (KP), dargestellt durch

$$KP_r = \frac{\sum_{n=1}^{r} \sum_{\tau=-(N-1)}^{N-1} c_\tau \cdot \exp(2\pi i \lambda_n \tau)}{\sum_{n=1}^{N} \sum_{\tau=-(N-1)}^{N-1} c_\tau \cdot \exp(2\pi i \lambda_n \tau)} \tag{2}$$

als Teststatistik der Hypothese des weißen Rauschen basiert auf den Überlegungen in Abschnitt 2 und die Berechnung ist entlehnt an [5]. Da das Spektrum des weißen Rauschens konstant ist und KP_r den Anteil spektraler Masse der ersten r Fourierfreqeuenzen angibt, sollte das kumulierte Periodogramm um die Winkelhalbierende des Einheitsquadrates schwanken, wenn die Hypothese zutrifft. [5] Die verwendete Verlustfunktion erweitert sich somit zu:

$$L_{\text{Periodogramm}} = \alpha \cdot \beta \cdot \frac{1}{N} \sum_{r=1}^{N} KP_r + (1-\alpha) \cdot MSE \tag{3}$$

Wichtig ist dabei zu erwähnen, dass das Periodogramm zwar ein Erwartungstreuer, aber kein konsistenter Schätzer ist. Dies bedeutet, dass die Varianz des Periodogramms für eine steigende Anzahl betrachteter Zufallsvariablen nicht gegen Null konvergiert. In der Literatur werden mehrere Vorschläge unterbreitet dem entgegenzuwirken. Unter anderem wird in dieser Arbeit der Ansatz des Moving Average Filters von Bartlett verwendet, welcher für die Prognose die Walking Forward Validierung nutzt. [4, 6]

3.4 White Noise Based Loss

Eine Erweiterung des kumulierten Periodogramms stellt der White Noise Based Loss (WNBL) mittels

$$L_{\text{WNBL}} = \alpha \cdot \beta \cdot \frac{1}{N} \sum_{r=1}^{N} (KP_r(R) - KP_r(W)) + (1-\alpha) \cdot MSE \tag{4}$$

dar, der die Zielrichtung der Verlustfunktion durch die Abweichung des kumulierten Periodogramms der Restsequenz $KP(R)$ von einem kumulierten Periodogramm weißen Rauschens $KP(W)$ explizit vorgibt.

3.5 White Noise Signal Loss

Der White Noise Signal Loss (WNSL) ist die dritte vorgeschlagene Verlustfunktion, um die Gleichheit von R und weißem Rauschen festzustellen und definiert diese auf Signalebene. Die Gleichung

$$L_{\text{WNSL}} = \alpha \cdot \beta \cdot \frac{1}{N} \sum_{n=1}^{N} (r_n - w_n)^2 + (1-\alpha) \cdot MSE \tag{5}$$

optimiert den punktweisen quadrierten Fehler der Restsequenz R und dem Signal W.

4 Modellierung

Die Datengrundlage besteht aus einer plausiblen und vollständigen Zeitreihen für 3 Jahre mit einer zeitlichen Auflösung von 15 Minuten. Für alle Untersuchungsszenarien wird der Week-Ahead Prognosehorizont festgelegt, bei einer Eingangssequenzlänge von einem Monat, was einem Verhältnis von 2880 auf 672 Sequenzpunkten entspricht. Als Modell wird ein MLP verwendet mit 2 verdeckten Schichten mit jeweils 24 Neuronen. Die Aktivierungsfunktion ist ReLU und der eingesetzte Optimierer Adam. [5] Das MLP wurde ausgewählt, da es im Bereich des Deep Learning die Standard Netz- und Neuronenarchitektur darstellt und die Ergebnisse so besser in den bestehenden Forschungsstand eingeordnet werden können.

5 Simulative Untersuchung und Evaluierung

Ein Auszug der Ergebnisse der Untersuchung sind in Tabelle 1 dargestellt. Die Verlustfunktion des White Noise Based Loss und kumulierten Periodogramm

Tabelle 1: Methode der Verlustfunktion und Fehlermaß

Verlustfunktion	MSE	Periodogramm	WNSL	WNBL
RMSE	0.968	0.721	0.957	0.721
MAPE (%)	9.66	7.49	9.43	7.48
MAE	0.737	0.572	0.72	0.571
α	0	0.99	0.1	0.99

erzielte in allen gemessenen Fehlermaßen die besten Ergebnisse. Eine verbesserte Generalisierungsfähigkeit kann bei Untersuchung der Prognosegüte über die gesamte Sequenzausgabe festgestellt werden. Abbildung 1 zeigt den RMSE der MSE- und WNBL-Verlustfunktion, deutlich zu erkennen ist ein positiver Trend der MSE-Verlustfunktion über die Sequenzlänge, ebenso wie die Minima an den Autokorrelationsspitzen im Datensatz, den Tagesrythmen (96 Sequenzpunkte). Im Vergleich dazu nähert sich der Verlustverlauf des WNBL mehr dem Idealverlauf eines vollständig unabhängig zufälligen Messfehlerverlaufs (siehe Abbildung 1).
Der durchgeführte Vergleich von Methodiken für Verlustfunktionen für künstlicher neuronaler Netze zur Energielastprognose hat gezeigt, dass durch die Verwendung von erweiterten Verlustfunktionen auf Basis des weißen Rauschens die Prognosegüte und Konvergenz gesteigert werden kann.

6 Zusammenfassung und Ausblick

Im Ergebnis der Untersuchung konnte bestätigt werden, dass die verwendeten Varianten der Verlustfunktionen einen nachweisbaren positiven Effekt auf die Modellperformance hinsichtlich der Generalisierungsfähigkeit und Robustheit haben. Von allen drei untersuchten Verlustfunktionen, basierend auf weißem Rauschen, erzielte der White Noise Based Loss die besten Ergebnisse. Im direkten Vergleich weißes Rauschen basierter Verlustfunktionen zu bereits etablierten, wie dem MSE, ist ein erhöhter Justierungsaufwand durch den Einfluss- und Skalierungsfaktor gegeben, was dem Model einen zusätzlichen optimierbaren Hyperparameter hinzufügt

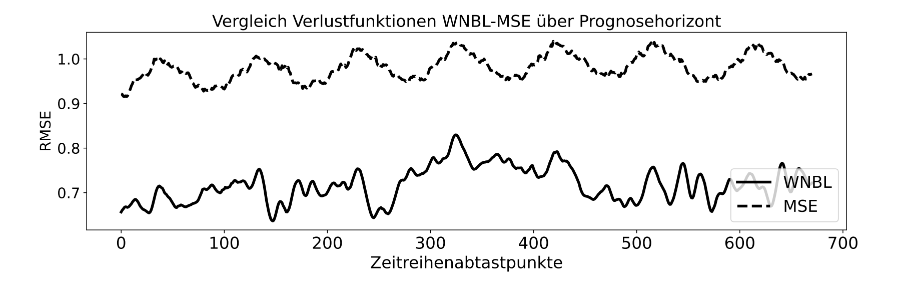


Bild 1: RMSE der MSE- und WNBL-Verlustfunktion

Literatur

[1] I. Goodfellow, Y. Bengio und A. Courville. „Deep Learning: Das umfassende Handbuch: Grundlagen, aktuelle Verfahren und Algorithmen, neue Forschungsansätze“. Frechen: mtip. 1. Auflage. 2018.

[2] L. Fahrmeir, T. Kneib und S. Lang. „Regression: Modelle, Methoden und Anwendungen“. Berlin, Heidelberg: Springer Berlin Heidelberg. 2. Auflage 2009.

[3] O. D. Doleski. „Herausforderung Utility 4.0: Wie sich die Energiewirtschaft im Zeitalter der Digitalisierung verändert“. Wiesbaden: Springer Vieweg. 2017.

[4] J.-P. Kreiß und G. Neuhaus. „Einführung in die Zeitreihenanalyse“. Berlin, Heidelberg: Springer Berlin Heidelberg. 2006.

[5] R. Schlittgen und B. H. J. Streitberg. „Zeitreihenanalyse“. München, Oldenbourg: Oldenbourg. 9. Auflage. 2011.

[6] L. H. Koopmans. „The Spectral Analysis of Time Series“. Erscheinungsort nicht ermittelbar: Elsevier Science & Technology. 2. Auflage. 2011.

[7] Z. Wang und A. C. Bovik. „Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures“. In: IEEE Signal Process. Mag., Jg. 26, Nr. 1, S. 98-117. 2009.

[8] X. Cui, W. Zhang, Z. Tüske und M. Picheny. „Evolutionary Stochastic Gradient Descent for Optimization of Deep Neural Networks“. arXiv-preprint. 1810.06773v1. 2018.

[9] D. P. Kingma und J. Ba. „Adam: A Method for Stochastic Optimization“. arXiv-preprint. arXiv:1412.6980v9. 2014.

Autonomes Putten mittels datengetriebener und physikbasierter Methoden

Annika Junker, Niklas Fittkau, Julia Timmermann, Ansgar Trächtler

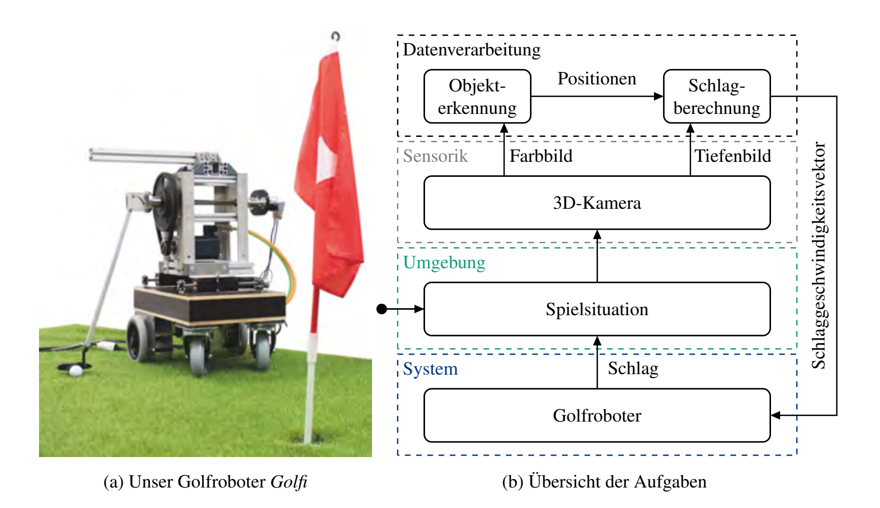
Heinz Nixdorf Institut, Universität Paderborn
Fürstenallee 11, 33102 Paderborn
E-Mail: annika.junker@hni.upb.de

1 Einleitung

Im Bereich des Golfsports werden bereits Roboter für das Training von Golfspieler*innen [1] sowie zum Testen der Schläger [2, 3, 4, 5, 6] eingesetzt. Der Roboter *ROB-OT* [7], der von einem*einer Golfspieler*in gesteuert wird, kann Bälle schlagen und wird zumeist zu Unterhaltungszwecken eingesetzt.

Unser Golfroboter *Golfi*, siehe Bild 1a, verfolgt das Ziel, vollständig autonom zu putten, d. h. den Ball nach einer Trainingsphase von einer beliebigen Startposition aus mit nur einem Schlag in das Loch zu schlagen. Die mechatronische Ansteuerung des Roboters lässt sich geradlinig durch klassische regelungstechnische Methoden realisieren, wohingegen sich die Analyse der Spielsituation und die Entwicklung einer Spielstrategie deutlich anspruchsvoller gestalten. Daher nutzen wir eine synergetische Kombination aus leistungsstarken datengetriebenen und bewährten physikbasierten Methoden aus dem regelungstechnischen Kontext. Die unterschiedlichen Aufgaben sind in Bild 1b dargestellt, wobei die Komplexität von unten nach oben zunimmt.

In diesem Kurzbeitrag beschreiben wir unsere Strategie für die Schlagberechnung mittels eines mit physikalischem Vorwissen trainierten neuronalen Netzes. [8] liefert eine detaillierte Beschreibung des mechatronischen Entwurfs bestehend aus Fahreinheit und Schlagregelung sowie der Bildverarbeitung.

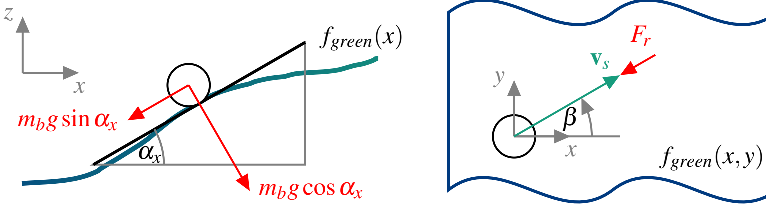


(a) Unser Golfroboter *Golfi* (b) Übersicht der Aufgaben

Bild 1: *Golfi* ist ein selbstlernender Roboter zum autonomen Putten. Die unterschiedlichen Aufgaben werden physikbasiert, datengetrieben oder mittels einer Kombination beider Methoden gelöst. Die 3D-Kamera ist der Einfachheit halber an der Zimmerdecke des Labors befestigt und aus der Vogelperspektive auf die Spielfläche ausgerichtet.

2 Strategie zur Bestimmung des optimalen Schlaggeschwindigkeitsvektors

Die Bestimmung des optimalen Schlaggeschwindigkeitsvektors erfolgt mittels eines neuronalen Netzes, das die Dynamik des Golfballs für eine gegebene hügelige Spielfläche abbildet. Um die Anzahl der (zeit-)aufwändigen Interaktionen mit dem realen System zu reduzieren, werden die Trainingsdaten simulativ anhand eines physikalischen Modells erzeugt. Für eine konkrete Spielsituation wird dann anhand des trainierten neuronalen Netzes der erforderliche Schlaggeschwindigkeitsvektor berechnet, sodass der Ball in das Loch rollt. Falls der Ball nach der Ausführung des Schlags nicht in das Loch rollt, ist es denkbar, diesen Schlag als zusätzlichen realen Trainingsschlag zum Nachtrainieren des neuronalen Netzes zu verwenden.



(a) Betrachtung als schiefe Ebene

(b) Rollrichtung und Reibkraft des Balls

Bild 2: Das physikalische Modell der Golfballdynamik basiert auf schiefen Ebenen. Die Reibkraft F_r wirkt parallel zur Oberfläche und entgegengesetzt zur Rollrichtung des Balls.

2.1 Simulativ erzeugte Trainingsschläge

Die Trainingsschläge ergeben sich aus der Simulation eines physikbasierten Modells der Golfballdynamik. Die Spielfläche wird aus dem Tiefenbild der 3D-Kamera als Fläche $f_{green}(x,y)$ approximiert und lokal als schiefe Ebenen mit den Winkeln

$$\alpha_x = \arctan\left(\frac{\partial f_{green}(x,y)}{\partial x}\right), \alpha_y = \arctan\left(\frac{\partial f_{green}(x,y)}{\partial y}\right). \tag{1}$$

aufgefasst, vgl. Bild 2a. Der Rollwiderstand $F_r = m_b g \mu_b \cos\alpha_x \cos\alpha_y$ wird als konstant [9] und mit Wirkrichtung entgegen der Rollrichtung $\beta = \arctan(\dot{y}/\dot{x})$ des Balles angenommen, vgl. Bild 2b. m_b ist die Masse des Balls, g die Gravitationskonstante und μ_b der Rollwiderstandskoeffizient. Damit ergibt sich

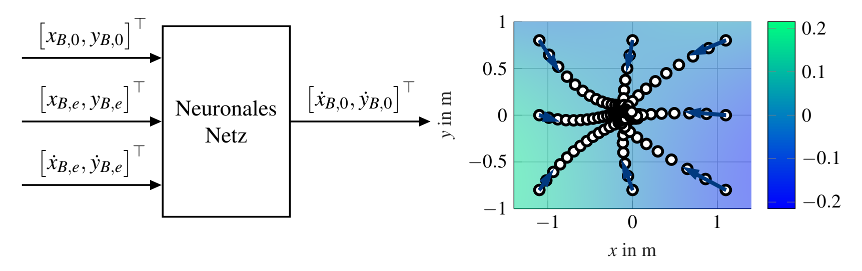
$$m_b\ddot{x} = -m_b g \sin\alpha_x - F_r|\cos\beta|\operatorname{sgn}(\dot{x}), \tag{2}$$

$$m_b\ddot{y} = -m_b g \sin\alpha_y - F_r|\sin\beta|\operatorname{sgn}(\dot{y}), \tag{3}$$

Für das Training werden zufällige Schläge mit dem Balldynamikmodell (2)-(3) mittels RK4-Solver simuliert.

2.2 Architektur des neuronalen Netzes

Das neuronale Netz hat die Aufgabe, den optimalen Schlaggeschwindigkeitsvektor $\left[\dot{x}_{B,0}, \dot{y}_{B,0}\right]^\top$ für eine initiale Ballposition $\left[x_{B,0}, y_{B,0}\right]^\top$ so zu bestimmen,



(a) Architektur des neuronalen Netzes

(b) Ergebnis für eine fiktive Spielfläche mit Hügeln

Bild 3: Bestimmung des optimalen Schlaggeschwindigkeitsvektors.

dass der Ball im Loch mit der Position $[x_H, y_H]^\top$ liegen bleibt, d. h.

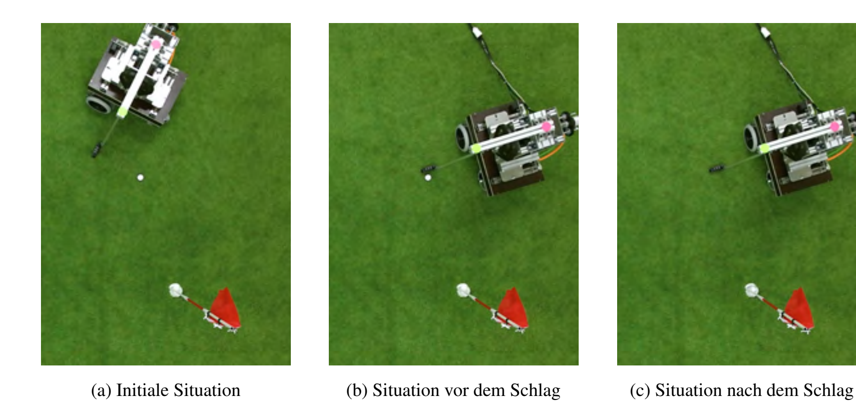
$$\begin{bmatrix} x_{B,e} \\ y_{B,e} \end{bmatrix} = \begin{bmatrix} x_H \\ y_H \end{bmatrix}, \begin{bmatrix} \dot{x}_{B,e} \\ \dot{y}_{B,e} \end{bmatrix} = \begin{bmatrix} \dot{x}_H \\ \dot{y}_H \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \tag{4}$$

wobei der Index e die Endposition des Balles kennzeichnet. Das für diese Aufgabe entwickelte neuronale Netz mit zwei Schichten und jeweils 30 versteckten Neuronen (vgl. Bild 3a) wird mit den Trainingsdaten aus Abschnitt 2.1 trainiert. Für eine konkrete Spielsituation mit gegebener Ball- und Lochposition bestimmt das Netz dann mittels (4) explizit den erforderlichen Schlaggeschwindigkeitsvektor.

3 Ergebnisse und Ausblick

Bisher haben wir das autonome Golfspiel für eine Spielfläche ohne Hügel umgesetzt. Für eine konkrete Spielsituation werden zunächst die Positionen des Golfroboters und des Balls detektiert (vgl. Bild 4a). Danach werden der optimale Schlaggeschwindigkeitsvektor und die damit einhergehende Zielpose des Golfroboters berechnet. Schließlich wird der Golfroboter so angesteuert, dass er die Zielpose einnimmt (vgl. Bild 4b) und den Ball in das Loch schlägt (vgl. Bild 4c).

Unsere Arbeit demonstriert, wie sich datengetriebene und physikbasierte Methoden vorteilhaft kombinieren lassen. Durch ein einfaches physikalisches Mo-



(a) Initiale Situation (b) Situation vor dem Schlag (c) Situation nach dem Schlag

Bild 4: Beispielhafter Ablauf einer konkreten Spielsituation für eine Spielfläche ohne Hügel. Als Trainingsdaten wurden 3000 zufällige simulierte Schläge verwendet.

dell der Balldynamik lässt sich ein neuronales Netz so trainieren, dass es zuverlässig berechnet, wie der Ball geschlagen werden muss, damit er in das Loch rollt.

Modellbasiert konnten wir die Machbarkeit unserer Strategie bereits auch für Spielflächen mit Hügeln zeigen (vgl. Bild 3b). Mit den vom neuronalen Netz bestimmten Schlaggeschwindigkeitsvektoren (dunkelblau) rollt der Ball plausibel in Richtung des Lochs im Ursprung. Eine Validierung am realen System steht für das Szenario mit Hügeln noch aus. Weitere Forschung sollte außerdem darauf abzielen, die Möglichkeit des Nachtrainierens in den Prozess zu integrieren. Dabei stellt sich die Frage, wie die realen Trainingsschläge einbezogen werden können, z. B. in Bezug auf die Gewichtung im Vergleich zu den simulativ erzeugten Trainingsschlägen.

4 Danksagung

Das diesem Bericht zugrunde liegende Vorhaben wurde mit Mitteln des Bundesministeriums für Bildung und Forschung unter dem Förderkennzeichen 01IS20052 gefördert. Die Verantwortung für den lnhalt dieser Veröffentlichung liegt bei den Autor*innen.

Literatur

[1] RoboGolfPro. (2017) Robo golfpro. [Online]. Available: https://robogolfpro.com/

[2] N. Betzler, “The effect of differing shaft dynamics on the biomechanics of the golf swing,” Ph.D. dissertation, Edinburgh Napier University, Edinburgh, 2010.

[3] C. Cogswell, R. Hathaway, and J. Voinovich, “Golf putter testing mechanism,” Ph.D. dissertation, California Polytechnic State University, San Luis Obispo, 2016.

[4] Golf Tips Magazine. (2003) Become a swing machine. [Online]. Available: https://www.golftipsmag.com/instruction/faults-and-fixes/become-a-swing-machine/

[5] The Irish Times. (1998) The last swing of iron byron. [Online]. Available: https://www.irishtimes.com/sport/the-last-swing-of-iron-byron-1.195958

[6] Miyamae Co.,ltd. Miya robo-10: Swing robot with rotating body, robo 10 from miyamae. [Online]. Available: http://www.miyamae.co.jp/english/golf/img/robo10.pdf

[7] The Golf Wire. (2018) Rob-ot makes an appearance at the america’s warrior partnership golf tournament in la quinta, ca. [Online]. Available: https://thegolfwire.com/rob-ot-makes-appearance-americas-warrior-partnership-golf-tournament-la-quinta-ca/

[8] A. Junker, N. Fittkau, J. Timmermann, and A. Trächtler, “Autonomous golf putting with data-driven and physics-based methods” (eingereicht), 2022.

[9] Transportation Research Board, *Tires and Passenger Vehicle Fuel Economy: Informing Consumers, Improving Performance – Special Report 286*. Washington, DC: The National Academies Press, 2006.

Conditional Behavior Prediction for Automated Driving on Highways

Christopher Diehl[1], Timo Osterburg[1], Nils Murzyn[2], Georg Schneider[2], Frank Hoffmann[1], Torsten Bertram[1]

[1] Institute of Control Theory and Systems Engineering, TU Dortmund
44227 Dortmund, Germany
E-Mail: forename.surname@tu-dortmunde.de

[2] ZF Friedrichshafen AG, Artificial Intelligence Lab,
66123 Saarbrücken, Germany
E-Mail: forename.surname@zf.com

1 Introduction

Predicting the future behavior in multi-agent systems is critical for safe motion planning and control of self-driving vehicles (SDV). Here, prediction models must account for the static geometry (e.g., static obstacles and high-definition (HD) map data) of the environment and the interactions between different moving objects. Deep Learning architectures achieve state-of-the-art results on many forecasting benchmarks. Initially, methods like [3] describe the scene as a birds-eye-view image, coming with discretization inaccuracies, high memory consumption, and computational complexity. Recent approaches [2, 7, 9] describe the traffic scene as a graph. While improving the performance, standard models predict without considering a drivers intent. In contrast, [4] and [5] conditions the prediction on a single planned trajectory or the action of the SDV, which has the downside of possible overly confident anticipation of how the SDV may influence the other agents' behavior [8]. [6] and [10] condition on fixed goal states and [7] relies on a a high-level route in urban environments. This work focuses on the highway environment and wants to answer the ques-

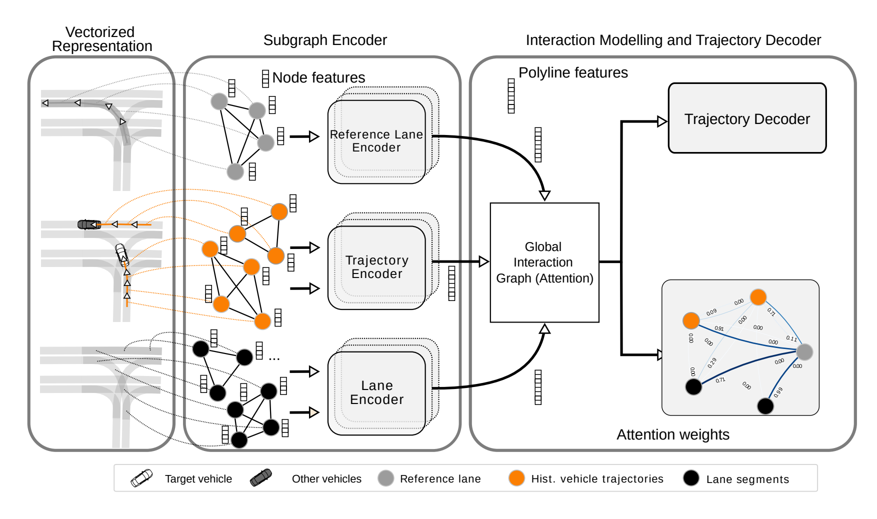


Figure 1: System architecture of conditional behavior prediction.

tion: *Do trajectory prediction methods improve by conditioning on high-level information in highway scenarios.*

2 Conditional Behavior Prediction

Problem Formulation. Assume a multi-agent system with $N+1$ traffic participants (one target agent and $N \in \mathbb{N}^+$ other agents) described by discrete time steps t. $\mathbf{x}_t = [x_t, y_t]^\top$ describes one agent's past state and $\mathbf{y}_t = [x_t, y_t]^\top$ future state. $\mathbf{x}^n$ and $\mathbf{y}^n$ describe the sequence of past and future states of agent indexed by $n \in N+1$ respectively, whereas the history has $H \in \mathbb{N}^+$ time steps and the future $F \in \mathbb{N}^+$ time steps. $n = 0$ denotes the index of the target agent. Let $\mathbf{X} = (\mathbf{x}^0, \mathbf{x}^1, \ldots, \mathbf{x}^N)$ denote the joint historic state sequence of all agents, $\mathbf{M} \in \mathbb{R}^{n_\mathrm{M}}$ an HD map and $\mathbf{R} \in \mathbb{R}^{n_\mathrm{R}}$ the high-level route information with dimensions n_M and n_R. We could then model the probability density function of future target agent states with $p_\theta(\mathbf{y^0}|\mathbf{X}, \mathbf{M}, \mathbf{R})$. Our goal is to learn the parameters $\theta \in \mathbb{R}^{n_\theta}$ with dimension n_θ of a neural network under a maximum likelihood estimation.

Approach. Similar to [11], the behavior prediction problem relies on a graph-based state representation shown in figure 1. Agent trajectories, lane information and high-level route are represented as polylines. Whereas driving lanes are represented by their left and right lane boundaries, the high-level route is captured by the lane centerline. Polylines $P_j \in \mathscr{P}$ with index $j \in \mathbb{N}^+$ are mapped onto $m-1$ equidistant vectors $\mathbf{v}_i \in P_j$ with $\mathbf{v}_i = \left[(\mathbf{d}_i^s)^\mathrm{T}, (\mathbf{d}_i^\mathrm{e})^\mathrm{T}, (\mathbf{a}_i)^\mathrm{T}, j\right]^\mathrm{T}$. $\mathbf{d}_i^\mathrm{s}, \mathbf{d}_i^\mathrm{e} \in \mathbb{R}^2$ denote the 2-D start and end positions w.r.t. the self-driving vehicles coordinate system with $i \in \mathbb{N}^+$. Further, $\mathbf{a}_i \in \mathbb{R}^{n_\mathrm{a}}$ is a set of polyline attributes with dimension n_a. A one hot vector classifies the lane type (route or normal lane polyline). In VectorNet (VN) [2], fully connected sub-graphs encode the corresponding information by multi-layer-perceptrons (MLP). A self-attention mechanism captures the higher-order interactions between sub-graphs and provides attention weights. Inspection of the attention weights provides inside into the relevance of polylines for the agents decision making. The trajectory is then decoded using another MLP. In the unimodal case, the network predicts a single trajectory that minimizes the L2 Loss between predicted and ground truth trajectory in the training data. This work also extends VN to predict k trajectories. In the multi-modal case, the approach minimizes the min-of-k loss [3]. C-VN denotes our conditional model.

3 Evaluation

This section evaluates the approach in highway scenarios with dense traffic in the CARLA [13] simulator (Version 0.9.11) by comparing C-VN to the unconditioned model (VN). The simulated training data is generated by the CARLA traffic manager mode that controls the behavior of all agents simultaneously. The lane change (LC) behavior of the target agent was modified using the LC model of [12].

Metrics. *Average Displacement Error* (ADE): The displacement error between the predicted trajectory and the ground truth averaged over all time steps for unimodal prediction. *Final Displacement Error* (FDE): The displacement error between the predicted trajectory and the ground truth averaged over the last

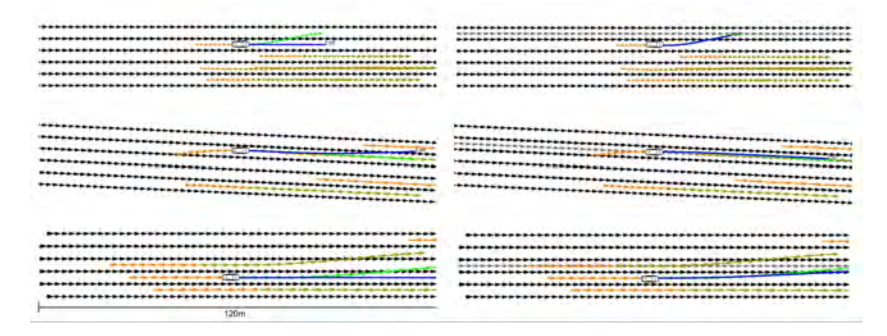


Figure 2: Qualitative comparison. Left: VN. Right: C-VN. All elements are visualized as vectors. Black: Lane Markings. Grey: Center line of the target lane. Orange: Agent position history. Blue: Predicted trajectory. Green: Ground truth future trajectory of the target agent. Yellow: Ground truth future trajectory of other agents.

time steps of each prediction, when the model only predicts one mode. *Minimum Average Displacement Error* ($\min_k$ADE): In the multimodal case, the model predicts k trajectories and the evaluation calculates the ADE for every mode and then takes the minimum.

Prediction Results. Figure 2 compares prediction results using VN or C-VN. Notice that the additional high-level information allows the model to predict lane changes earlier (rows one and three). Row two further shows that prediction better matches the ground truth and predicts a trajectory that terminates closer to the target lane center. The quantitative results in Table 1 support the initial hypothesis that conditioning benefits the trajectory prediction.

Applications to Motion Planning. In the framework of imitation learning the same approach can be employed for motion planning of an SDV rather than prediction. In that case, the vehicle is regulated along the predicted trajectory by underlying PID controllers for lateral and longitudinal motion. Figure 3 visualizes the planned trajectory during successful closed-loop control in a dense lane change scenario.

Table 1: Quantitative Prediction results. Bold numbers mark the best result.

Metric	ADE [m] ↓	FDE [m] ↓	$\min_2$ADE [m] ↓
VN	0.60	1.77	0.52
C-VN	**0.58**	**1.60**	**0.49**

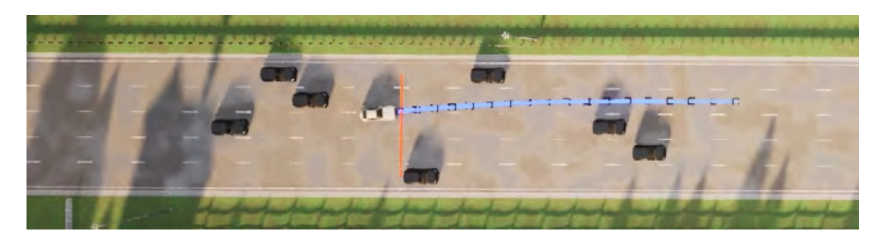


Figure 3: Results in CARLA by controlling a SDV by C-VN as a motion planning module. The blue visualizes the predicted trajectory of the SDV (white). The SDV performs a lane change in dense traffic triggered shortly before the orange line.

4 Conclusion

This work presents a graph-based conditional behavior prediction approach for automated driving. The comparative analysis reveals that the conditioning improves the prediction. We further demonstrate the suitability of conditional behavior prediction for motion planning. Future work should investigate the conditioning on network predictions of agents joint behaviors.

Acknowledgments

This research was funded by the Federal Ministry for Economic Affairs and Energy on the basis of a decision by the German Bundestag in the project ”KISSaF - AI-based Situation Interpretation for Automated Driving”.

References

[1] A. Vaswani et al. “Attention is All you Need”. In *Proc. of the Advances in Neural Information Processing Systems (NIPS)*. 2017.

[2] J. Gao et al. “VectorNet: Encoding hd maps and agent dynamics from vectorized representation”. In: *Proc. of the Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[3] H. Cui et al. “Multimodal Trajectory Predictions for Autonomous Driving using Deep Convolutional Networks”. In: *International Conference on Robotics and Automation (ICRA)*. 2019.

[4] T. Salzmann et al. “Trajectron++: Dynamically-Feasible Trajectory Forecasting With Heterogeneous Data”. In: *European Conference on Computer Vision (ECCV)*. 2020.

[5] C. Diehl, T. Sievernich, M. Krüger, F. Hoffman, T. Bertram “UMBRELLA: Uncertainty-Aware Model-Based Offline Reinforcement Learning Leveraging Planning”. In: *Advances in Neural Information Processing Systems, Machine Learning for Autonomous Driving Workshop*. 2021.

[6] N. Rhinehardt, R. McAllister, K. Kitani, S. Levine “PRECOG: PREdiction Conditioned On Goals in Visual Multi-Agent Settings”. In: *International Conference on Computer Vision (ICCV)*. 2019.

[7] S. Khandelwal et al. “What-if motion prediction for autonomous driving”. In: *arXiv preprint arXiv:2008.10587*. 2020.

[8] S. Tang, W. Zhan, M. Tomizuka “Interventional Behavior Prediction: Avoiding Overly Confident Anticipation in Interactive Prediction”. In: *arXiv preprint arXiv:2204.08665*. 2022.

[9] M. Liang et al. “Learning Lane Graph Representations for Motion Forecasting”. In: *European Conference on Computer Vision (ECCV)*. 2020.

[10] J. Ngiam et al. "Scene Transformer: A unified architecture for predicting multiple agent trajectories". In: *International Conference on Learning Representations (ICLR)*. 2022.

[11] C. Diehl, T. Waldeyer, F. Hoffman T. Bertram "VectorRL: Interpretable Graph-based Reinforcement Learning for Automated Driving". In: *31. Workshop Computational Intelligence.*. 2021.

[12] C. Wissing, K.-H. Glander, C. Haß, T. Nattermann, T. Bertram "Development and test of a lane change prediction algorithm for automated driving". In: *Fahrerassistenzsysteme*. 2017.

[13] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. "CARLA: An open urban driving simulator". In *Proc. of the Conference on Robot Learning (CORL)*. 2017.

Eine Python-Toolbox zur datengetriebenen Modellierung von Spritzgießprozessen und Lösung von Optimalsteuerungsproblemen zur Steuerung der Bauteilqualität

Alexander Rehmer[1], Andreas Kroll[1]

[1]Fachgebiet Mess- und Regelungstechnik, Universität Kassel
Mönchebergstr. 7, 34125 Kassel
E-Mail: {alexander.rehmer,andreas.kroll}@mrt.uni-kassel.de

Kurzfassung

Eine Voraussetzung für die Realisierung weitgehend autonomer Fertigungssysteme ist das Vorhandensein von mathematischen Modellen, mit denen das Verhalten einer Produktionsanlage mit ausreichend hoher Genauigkeit prädiziert werden kann. Solche Modelle können in modellbasierten Optimierungsverfahren eingesetzt werden, deren Ergebnis zur Unterstützung des Anlagenfahrers oder zur Selbstoptimierung der Produktionsanlage verwendet werden kann. Produktionsprozesse weisen hoft hochgradig dynamisches und nichtlineares Verhalten auf, sodass eine Modellbildung für Anwender nicht zu bewältigen ist. In diesem Beitrag wird eine Toolbox zur Unterstützung bei der Modellierung und modellbasierten Prozessoptimierung eines Produktionsverfahrens, dem Kunststoff-Spritzgießprozess, vorgestellt.

1 Einführung

Die Digital Twin of Injection Molding (DIM) Toolbox wurde mit Python 3.8 entwickelt und getestet. Sie stellt grundsätzlich einen Wrapper für das CasADi-Framework dar, einer Open-Source Bibliothek für nichtlineare Optimierung

und algorithmische Differentation [1]. Die prototypische DIM-Toolbox stellt Klassen und Funktionen bereit, welche die spezifischen Anforderungen bei der Modellbildung des Spritzgießprozesses berücksichtigt. Darüber hinaus werden numerische Optimalsteuerungsmethoden zur modellbasierten Prozessoptimierung bereitgestellt. Die Toolbox wird unter der BSD-Lizenz vertrieben und kann auf GitHub[1] heruntergeladen werden.

1.1 Digitale Zwillinge in Produktionsprozessen

Durch die im Kontext von Industrie 4.0 und dem Internet of Things (IoT) voranschreitenden Entwicklungen auf dem Gebiet der Automatisierung und Digitalisierung stehen erstmals umfassende und zeitlich hochaufgelöste Daten von physischen Produktionssystemen (PPS) wie dem Kunststoff-Spritzgießen zentral zur Verfügung. Dies ermöglicht die datengetriebene Bildung eines statischen oder dynamischen Modells, d.h. eines Digitalen Zwillings (DT), welcher in Verbindung mit dem physischen Produktionssystem ein cyber-physisches Produktionssystem bildet (CPPS). In Verbindung mit modellbasierten Regelungs- und Optimierungsverfahren kann der Digitale Zwilling zur Optimierung des Produktionsprozesses, meist hinsichtlich der Bauteilqualität, eingesetzt werden, und ist damit grundlegend für die Realisierung eines intelligenten Fertigungssystems [2].

Herausforderungen hierbei sind zum einen die Bildung eines Digitalen Zwillings, insbesondere falls eine dynamische Modellierung eines komplexen Produktionsprozesses wie dem Spritzgießen angestrebt wird. Zum anderen muss der Digitale Zwilling in das CPPS integriert werden. Während für die software- und datenseitige Integration des DT bereits Referenzarchitekturen entwickelt wurden [2, 3, 4], existieren keine Softwarebibliotheken die den Anwender bei der eigentlichen Bildung eines DT unterstützen sowie Methoden zur modellbasierten Prozessoptimierung bereitstellen. In diesem Beitrag wird daher eine im Rahmen des Projekts Digital Twin of Injection Molding[2] (DIM) entwickelte Toolbox zur datengetriebenen Modellbildung und modellbasierten Optimierung des Spritzgießprozesses vorgestellt.

[1] `https://github.com/MRT-RT/DigitalTwinInjectionMolding.git`
[2] `https://www.uni-kassel.de/go/DIM`



Bild 1: Spritzgießmaschine Arburg 470S. *Quelle: Fachgebiet Kunststofftechnik, IfW, Universität Kassel.*

1.2 Kunststoff-Spritzgießen

Das Spritzgießen ist ein urformendes Fertigungsverfahren bei dem plastifizierter Kunststoff unter Druck in eine Form, das Spritzgießwerkzeug, eingespritzt wird. Bild 1 zeigt die im Rahmen des Projektes verwendete Spritzgießmaschine. Der Prozess lässt sich üblicherweise in drei Phasen unterscheiden:

Einspritzphase: Eine definierte Schmelzmenge wird durch eine translatorische Bewegung der Schnecke in die Form injiziert. Diese Phase ist meist geschwindigkeitsgeregelt.

Nachdruckphase: Um den Materialschwund durch das Abkühlen der Schmelze in der Form zu kompensieren, wird ein definiertes Druckprofil aufgebracht. In dieser Phase wird meist der von der Maschine generierte hydraulische Druck geregelt.

Abkühlphase: In der drucklosen Abkühlphase soll das Bauteil soweit abkühlen, dass es sich deformationslos Entformen lässt.

Industrielle Spritzgießmaschinen verfügen meist über maschineninterne Regler, deren Parameter für den Nutzer unzugänglich sind. Darüber hinaus kann der Nutzer oft keine Sollwerttrajektorien $\{r_k\}_{k=0}^{T}$ vorgeben, sondern lediglich einige Maschinenparameter s, welche die Sollwerttrajektorie parametrieren. Hierbei sei k die diskrete Zeit und T die Dauer eines Spritzgießzyklus. Die gemessenen Prozessgrößen $\{p_k\}_{k=0}^{T}$ spiegeln den Verlauf des Prozesses wider und bestimmen auf unbekannte Weise die Eigenschaften Q des hergestellten

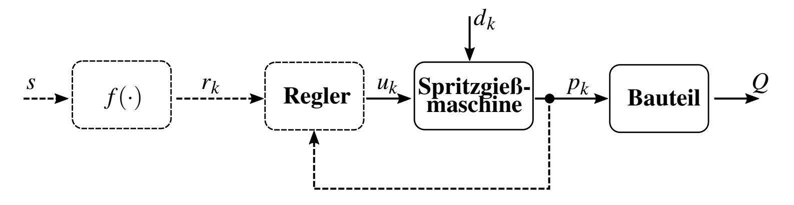


Bild 2: Schematische Darstellung des geregelten Spritzgießprozesses.

Bauteils. Häufig werden ausschließlich sogenannte maschinenseitige Prozessgrößen wie Geschwindigkeit v_k und Position der Schnecke x_k sowie Druck im Hydraulikaggregat p_k^{hyd} erfasst, da es sich hierbei um die Zielgrößen der maschineninternen Regler handelt. Entscheidend für die Bauteilqualität sind jedoch Prozessgrößen in der Kavität, bspw. der Werkzeuginnendruck p_k^{cav} und die Temperatur im Werkzeug T_k^{cav}. Bild 2 zeigt ein Blockschaltbild der gesamten Prozesskette.

2 Konzept der Modellbildung und Optimalsteuerung

Die Toolbox soll Anwender in Industrie und Praxis bei der Modellierung und Optimierung Ihrer Produktionsanlagen unterstützen. Produktionsanlagen unterscheiden sich oft bspw. hinsichtlich der implementierten Regelungskonzepte und verfügbaren Messgrößen. Eine Anforderung an die Toolbox ist somit ein modularer Aufbau, welcher dem Anwender erlaubt nur die für ihn sinnvollen Funktionalitäten zu nutzen.

Die in Bild 3 dargestellten von der Toolbox bereitgestellten wesentlichen Funktionalitäten sind

- Dynamische Qualitätsmodelle: Die Bild von Prozessgrößentrajektorien $\{p_k\}_{k=0}^{T}$ auf ein einzelnes Qualitätsdatum Q nach Ablauf der diskreten Zykluszeit T wird durch Modellstrukturen mit interner Dynamik realisiert. Diese Modelle können verwendet werden, um den optimalen Prozessgrößenverlauf für eine geforderte Bauteilqualität Q zu ermitteln.

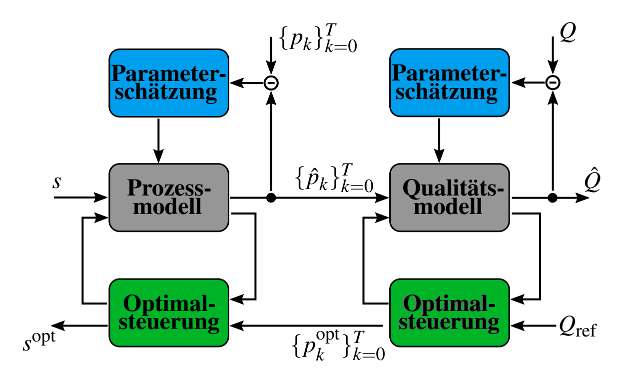


Bild 3: Modellierungs- und Optimierungskonzept.

- Dynamische Prozessmodelle: Ermöglichen die Bild von Stellgrößentrajektorien $\{u_k\}_{k=0}^{T}$ oder Maschinenparametern s_k auf die resultierenden Prozessgrößentrajektorien $\{p_k\}_{k=0}^{T}$.
- Parameteroptimierung: Methoden zur online und offline Parameteroptimierung der verwendeten Modellstrukturen.
- Optimalsteuerungsmethoden: Auf den Spritzgießprozess zugeschnittene Methoden zur Lösung von Optimalsteuerungsproblemen.

3 Toolbox

3.1 Datenstruktur

Um Kompatibilität zwischen allen Klassen zu gewährleisten, ist ein einheitliches Datenformat erforderlich. Für numerische und sequenzielle Daten existieren bereits etablierte Datentypen. Daher wurde beschlossen als einheitliches Datenformat für die Klassen der DIM-Toolbox ein Dictionary zu verwenden, in welchem die Daten in einer definierten Weise gespeichert werden. Die im einzelnen verwendeten Datentypen sind Listen, der `DataFrame`-Datentyp der `pandas`-Bibliothek und der `ndarray`-Datentyp der `numpy`-Bibliothek. verwendet. Wie in Bild 4 dargestellt, werden die Daten in einem Dictionary mit den Keys `'io_data'`, `'init_state'` und `'switch'` organisiert:

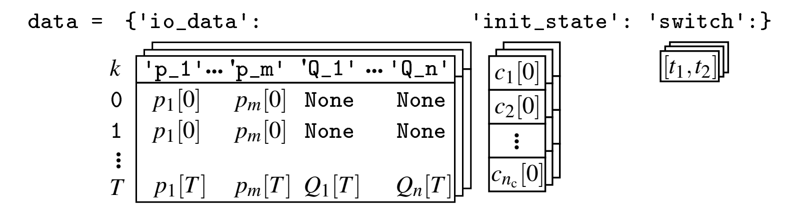


Bild 4: Standardisiertes Datenformat

`'io_data'`: Eine Liste von `pandas-DataFrames`. Jeder `DataFrame` enthält die mit einem Spritzgießzyklus korrespondierenden Prozessgrößen $\{p_k\}_{k=0}^{T}$ und Qualitätsgrößen Q_T. Da Q_T vektoriell ist, sind auch Werkzeuge mit mehreren Kavitäten abbildbar.

`'init_state'`: Eine Liste von `numpy-ndarrays`. Nur erforderlich für dynamische Modelle. Jedes Array repräsentiert den initialen Zustandsvektor des Modells zum Zyklusbeginn und hat eine der Modellordnung `dim_c`, siehe Bild 5, entsprechende Anzahl an Zeilen.

`'switch'`: Eine Liste von Listen, welche jeweils die diskreten Umschaltpunkte für den korrespondierenden Zyklus als `int` enthalten. Nur erforderlich, wenn mehrere zeitinvariante dynamische Modelle mittels einer der Wrapper-Klassen, siehe Bild 7, zu einem zeitvarianten schaltenden Modell verbunden werden. Auf diese Weise wird dem Modell mitgeteilt, zu welchen diskreten Zeitpunkten von einem zum nächsten Teilmodell umzuschalten ist.

3.2 Modellklassen

3.2.1 Basisklassen

Die Basisklasse für alle Modelle ist `Model`. `Model` stellt grundlegende Attribute, wie die Anzahl an Eingangsgrößen `dim_u` und Ausgangsgrößen `dim_y` sowie Funktionen, wie die Einschrittprädiktion `one_step_prediction()`, bereit, siehe Bild 5. Aus der `Model`-Klasse wurden wiederum zwei Klassen abgeleitet: `Dynamic` und `Static`. Die Klasse `Dynamic` stellt alle grundlegenden

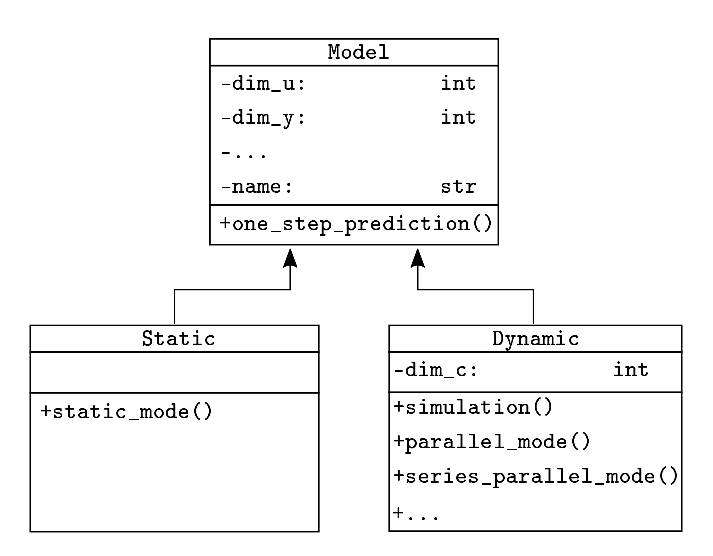


Bild 5: Basisklasse `Model` und daraus abgeleitete Klassen `Static` und `Dynamic`

Methoden eines dynamischen Modells bereit, d.h. die Auswertung in parallel Konfiguration `parallel_mode()` oder seriell-paralleler Konfiguration `series_parallel_mode()`. Die dynamische Ordnung des Modells wird durch das Attribut `dim_c` festgelegt. Sowohl interne Dynamikmodelle

$$\begin{aligned} x_{k+1} &= h\left(x_k, u_k\right) \\ y_k &= g\left(x_k\right) \end{aligned} \tag{1}$$

mit Eingangsgröße $boldsymbolu_k \in \mathbb{R}^{n_\mathrm{u}}$, Zustandsgröße $boldsymbolx_k \in \mathbb{R}^{n_\mathrm{x}}$ und Ausgangsgröße $boldsymboly_k \in \mathbb{R}^{n_\mathrm{y}}$ als auch externe Dynamikmodelle

$$y_k = f\left(y_{k-1}, \ldots, y_{k-m}, u_{k-1}, \ldots, u_{k-n}\right) \tag{2}$$

können aus dieser Basisklasse abgeleitet werden, siehe Bild 6. Die Modellgleichungen müssen durch den Nutzer in der Methode `Initialize()` der abgeleiteten Klasse als CasADi MXFunction definiert werden. Eine Reihe gängiger rekurrenter Modellstrukturen, wie die Gated Recurrent Unit (GRU) und das Long Short-Term Memory (LSTM) Netz, sind bereits vorimplementiert. Indem nur noch die Modellgleichungen nach einem vorgegebenen Schema

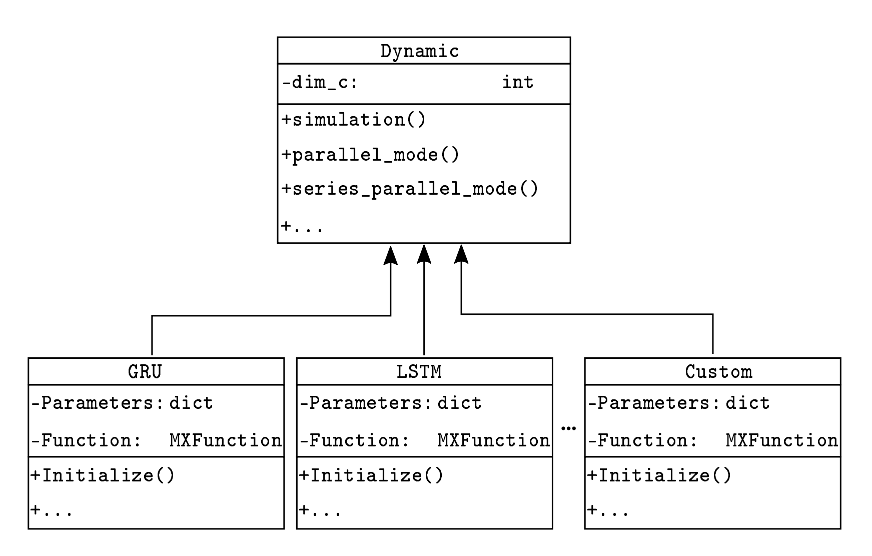


Bild 6: Basisklasse `Dynamic` und daraus abgeleitete Klassen

definiert werden müssen, soll dem Nutzer die die Implementierung eigener Modellansätze einfachst möglich gemacht werden.

Zudem wurden die Modellklassen so konstruiert, dass dasselbe Objekt vom Nutzer sowohl an Funktionen zur Parameterschätzung als auch zur Lösung numerischer Optimalsteuerungsprobleme übergeben werden kann. Hierfür werden bei der Definition der Modellgleichungen sowohl die Eingängsgrößen als auch die Modellparameter als symbolische Variablen definiert. Bei der Modellparameteroptimierung bzw. Optimierung der Eingangsgrößen werden dann die Eingangsgrößen bzw. Modellparameter automatisch von symbolischen in numerische Variablen umgewandelt. Aus der Basisklasse `Static` können statische Modellklassen, deren Gleichung vom Typ

$$y = f(u)$$

ist, abgeleitet werden. Verbreitete statische Modellstrukturen, wie Polynome und Multilayer-Perceptrons (MLP), sind bereits implementiert.

DynamicQualityModel	
-subsystems:	list
-name:	str
-...	
+Initialize()	
+Simulation()	
+...	

DynamicProcessModel	
-subsystems:	list
-name:	str
-...	
+Initialize()	
+Simulation()	
+...	

Bild 7: Wrapper-Klassen `DynamicQualityModel` und `DynamicProcessModel` zur Realisierung schaltender dynamischer Modelle.

3.2.2 Wrapper-Klassen

Wie in Abschnitt 1.2 erläutert, lässt sich der Spritzgießprozess in drei Phasen mit unterschiedlichem dynamischen Verhalten unterteilen. Bei einer dynamischen Modellierung wird dem Rechnung getragen, indem ein dediziertes Modell für jede Phase geschätzt wird. Das entstehende Gesamtmodell ist somit ein zeitvariantes schaltendes System bestehend aus drei zeitinvarianten Subsystemen $i = 1,2,3$ mit Umschaltpunkten t_1 und t_2:

$$\begin{aligned} x_{k+1} &= h^i\left(x_k, u_k; \Theta^i\right), \quad x_0 = 0, \quad k \in \mathbb{Z}^+, \quad i = \begin{cases} 1 \,\forall\, k \leq t_1, \\ 2 \,\forall\, t_1 < k \leq t_2, \\ 3 \,\forall\, k > t_2 \end{cases} \\ y_k &= g^i\left(x_k; \Phi^i\right) \end{aligned} \tag{3}$$

Damit ein aus mehreren Teilmodellen bestehendes Modell genauso ausgewertet, optimiert und für Optimalsteuerungsprobleme verwendet werden kann wie Modelle der Klasse `Dynamic()`, wird jeweils eine Wrapper-Klasse für Qualitätsmodelle `DynamicQualityModel` und für Prozessmodelle `DynamicProcessModel` bereitgestellt, siehe Bild 7. Die Teilmodelle vom Typ `Dynamic` werden der Wrapper-Klasse als Attribut `subsystems` übergeben. Die Teilmodelle werden dann mittels der `Simulation`-Methode in der richtigen Reihenfolge ausgewertet und die internen Modellzustände an den definierten Umschaltpunkten übergeben.

```
        ParamOptimizer
-model:              int
-data_train:         data
-data_val:           data
-...
+optimize()
-...
```

Bild 8: Klasse `ParamOptimizer` zur Optimierung der Parameter von Objekten der Klasse `Model`

3.3 Parameteroptimierung

Der Optimierer minimiert die Modellparameter θ so, dass die quadratische Kostenfunktion

$$\mathscr{L}(\theta) = \frac{1}{2}\left(\hat{y}_T(\theta) - y_T\right)^T \left(\hat{y}_T(\theta) - y_T\right) \tag{4}$$

minimiert wird. Bei der Erzeugung einer Instanz der Klasse `ParamOptimizer` wird eine Instanz der CasADi-Klasse `Opti` erstellt mithilfe derer das Optimierungsproblem formuliert wird. Die Modellparameter θ des zu optimierenden Modells `model` werden als Zielgrößen des Optimierungsproblems definiert. Durch Aufrufen der `optimize`-Methode wird das Modell (in Abhängigkeit der symbolischen Modellparameter) sowohl auf den Trainingsdaten `data_train` als auch auf den Validierungsdaten `data_val` ausgewertet. Die Parameterschätzung erfolgt auf den Trainingsdaten. Der Parametersatz, welcher während der Optimierung die geringsten Validierungskosten ergab, wird an den Nutzer zurückgegeben. Zur Parameterschätzung wird IPOPT [5] verwendet. Neben den zuvor beschriebenen grundlegenden Funktionalitäten, sind folgende zusätzliche Funktionalitäten in der Toolbox implementiert:

- Parallelisierung: Um die Initialisierungsabhängigkeit nichtlinearer Optimierungsprobleme zu berücksichtigen, werden oft Multistart-Strategien verfolgt. `ParamOptimizer` ermöglicht die parallele Optimierung mehrerer Initialisierungen.
- Online-Optimierung: Durch Übergabe von Optionen an den Solver - IPOPT können bspw. eine approximative Berechnung der Hesse-Matrix,

eine maximale Anzahl an Iterationen oder eine maximale Optimierungsdauer eingestellt werden.

- Randbedindungen: Es können Randbedingungen an den Optimierer übergeben werden.
- Parameter einfrieren: Parameter, die nicht optimiert werden sollen, können vom Nutzer eingefroren werden.

3.4 Numerische Optimalsteuerung

Wie aus Bild 3 hervorgeht, soll die Toolbox dem Nutzer die Möglichkeit bieten, zwei Optimalsteuerungsprobleme zu lösen:

- Die Ermittlung der optimalen einzustellenden Maschinenparameter s^{opt} gegeben die Referenztrajektorien für eine oder mehrere Prozessgrößen $\{p_k^{\mathrm{ref}}\}_{k=0}^{T}$ unter Verwendung eines dynamischen Prozessmodells.
- Die Ermittlung der optimalen Prozessgrößentrajektorien $\{p_k^{\mathrm{opt}}\}_{k=0}^{T}$ gegeben eine Referenz der zu realisierenden Bauteilqualität Q_{ref} unter Verwendung eines dynamischen Qualitätsmodells.

Für die Lösung des erstgenannten Optimierungsproblems stellt die Klasse `ProcessMultiStageOptimizer` eine auf den Spritzgießprozess zugeschnittene Variante des sogenannten Mehrfachschießverfahrens (*engl. direct multiple shooting*) bereit. Hinsichtlich des Spritzgießprozesses zu berücksichtigende Anforderungen umfassen insbesondere, dass zu definierten Zeitinstanzen zwischen Teilmodellen umgeschaltet wird. Sind die Eingangsgrößen des Prozessmodells zudem Sollwerttrajektorien für maschineninterne Regler, so können diese meist nicht beliebig vorgegeben werden, sondern sind Funktionen einiger weniger Maschinenparameter. `ProcessMultiStageOptimizer` erlaubt, die funktionale Abhängigkeit der Sollwerttrajektorien `reference_input` von den einstellbaren Maschinenparametern `ref_param` zu definieren. Somit sind dann nicht die gesamten Sollwerttrajektorien Gegenstand der Optimierung, sondern nur einige wenige Maschinenparameter.

```
ProcessMultiStageOptimizer
-------------------------------
-process_model:    Model
-target:           array
-reference_input:  function
-target:           dict
-...
-------------------------------
+optimize()
-...
```

```
QualityMultiStageOptimizer
-------------------------------
-quality_model:    Model
-target:           array
-...
-------------------------------
+optimize()
-...
```

Bild 9: Klassen `QualityMultiStageOptimizer` zur Lösung von Optimalsteuerungsproblemen mit einem Einschießverfahren und `ProcessMultiStageOptimizer` zur Lösung von Optimalsteuerungsproblemen mit einem Mehrfachschießverfahren.

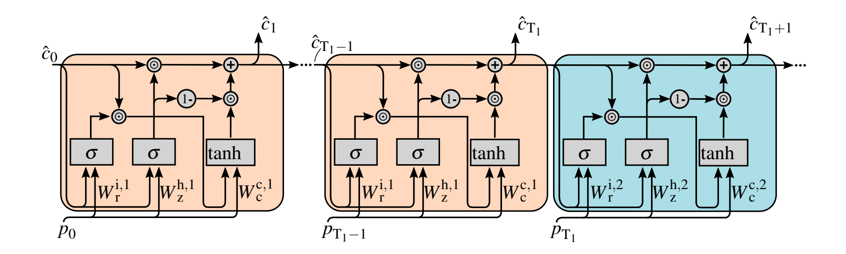
Die Ermittlung der optimalen Prozessgrößentrajektorien $\{p_k^{\mathrm{opt}}\}_{k=0}^{T}$ kann nicht mit einem Mehrfachschießverfahren gelöst werden, da für die Ausgangsgröße, d.h. die Bauteilqualität, keine Trajektorie sondern nur ein Endwert zur Verfügung steht. Daher wurde mit der Klasse `QualityMultiStageOptimizer`, siehe Bild 9, ein auf Einfachschießverfahren (*engl. direct single shooting*) basierender Löser implementiert. Das Umschalten zwischen Teilmodellen zu definierten Zeitpunkten wurde hier ebenfalls berücksichtigt sowie die Kostenfunktion an die Erreichung eines Endwertes angepasst.

4 Anwendungsbeispiel

In diesem Anwendungsbeispiel soll die Identifikation eines dynamischen Qualitätsmodells mittels der Toolbox demonstriert werden. Sowohl die Daten als auch der vollständige Code in Form eines Jupyter-Notebooks für diese Fallstudie sind in dem einleitend erwähnten GitHub-Repository verfügbar.

Aus umfassenden Voruntersuchungen [6] ist bekannt, dass eine Modellstruktur mit interner Dynamik, die zu guten Ergebnissen führt, die sogenannte Gated Recurrent Unit (GRU) ist. Die Zustandsgleichung dieser rekurrenten Netzarchitektur lautet:

$$\hat{c}_{k+1} = f_{\mathrm{z}} \odot \hat{c}_k + (1 - f_{\mathrm{z}}) \odot f_{\mathrm{c}}. \tag{5}$$



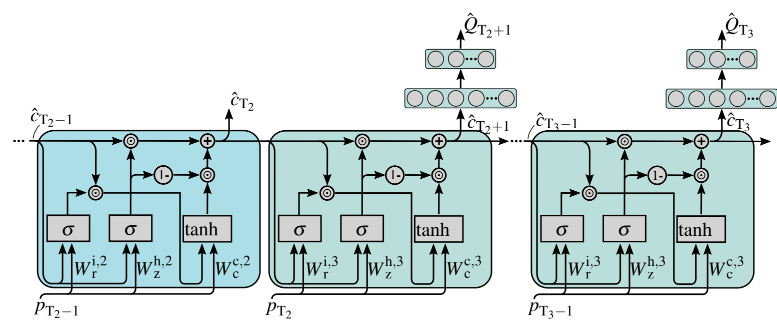


Bild 10: Zeitvariantes schaltendes Qualitätsmodell bestehend aus drei GRU Netzen und einem gewöhnlichen MLP als Ausgabegleichung.

Der Operator $\odot$ steht für das Hadamard-Produkt. Die Aktivierungen der sogenannten *Gates* f_{r}, f_{z} und f_{c} ergeben sich gemäß

$$
\begin{aligned}
f_{\mathrm{r}} &= \sigma\left(W_{\mathrm{r}} \cdot [\hat{c}_k, u_k]^T + b_{\mathrm{r}}\right), \\
f_{\mathrm{z}} &= \sigma\left(W_{\mathrm{z}} \cdot [\hat{c}_k, u_k]^T + b_{\mathrm{z}}\right), \\
f_{\mathrm{c}} &= \tanh\left(W_{\mathrm{c}} \cdot \left[\tilde{\hat{c}}_k, u_k\right]^T + b_{\mathrm{c}}\right),
\end{aligned}
\tag{6}
$$

mit $\tilde{\hat{c}}_k = f_{\mathrm{r}} \odot \hat{c}_k$.

Daher soll ein dynamisches Qualitätsmodell bestehend aus drei GRUs, jeweils für Einspritz-, Nachdruck- und Abkühlphase geschätzt werden, siehe Bild 10. Qualitätsrelevante Prozessgrößen sind die unmittelbar in der Form gemessene Temperatur `T_wkz_ist` und der Druck `p_wkz_ist`. Modelliert werden soll der Innendurchmesser `D_i` des produzierten Bauteils, [1] in Bild 11.



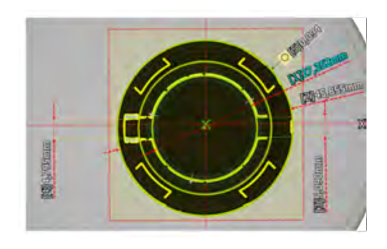


Bild 11: Foto des produzierten Bauteils (links) sowie Draufsicht mit optisch erfasster Bauteilgeometrie. *Quelle: Fachgebiet Kunststofftechnik, IfW, Universität Kassel.*

Die Bezeichnung der Ein- und Ausgangsgrößen des Modells muss mit den Spaltenbezeichnungen in den `DataFrames`, siehe Bild 4, übereinstimmen. Algorithmus 1 zeigt, wie das in Bild 10 dargestellte Modell mit der DIM-Toolbox generiert werden kann. Im Folgenden werden beispielhaft GRUs mit einem internen Modellzustand `dim_c=1` verwendet, `dim_c=1` entspricht n_x in (1). Die Ausgabegleichung, vergleiche (1), wird durch ein vorwärtsgerichtetes Neuronales Netz mit `dim_hidden=10` verdeckten Neuronen approximiert.

Algorithmus 1: Initialisierung des Modells in Bild 10

```
from DIM.models.model_structures import GRU
from DIM.models.injection_molding import DynamicQualityModel

u_label ← ['p_wkz_ist','T_wkz_ist']
y_label ← ['D_i']

options ← {'dim_u':2,'dim_c':1,'dim_hidden':10,
        'dim_out':1,'u_label':u_label,'y_label':y_label}

GRU_inj ← GRU(**options,name← 'GRU_inj' )
GRU_press ← GRU(**options,name← 'GRU_press' )
GRU_cool ← GRU(**options,name← 'GRU_cool' )

QModel ← DynamicQualityModel( subsystems← [GRU_inj,GRU_press,GRU_cool],
        name← 'QModel')
```

Die Parameteroptimierung erfolgt, indem das zeitvariante schaltende Qualitätsmodell zusammen mit Trainingsdaten `data_train` und Validierungsdaten `data_val`, welche wie in Abschnitt 3.1 beschrieben in einem Dictionary or-

ganisiert sind, an ein Objekt der Klasse `ParamOptimizer` übergeben werden und dessen `optimize`-Methode aufgerufen wird, siehe Algorithmus 2.

Algorithmus 2: Parameteroptimierung des Modells in Bild 10

```
import pickle
from DIM.optim.param_optim import ParamOptimizer

data_train, data_val ← pkl.load(open('data.pkl'))

param_optimizer ← ParamOptimizer(model← QModel,
        data_train← data_train,data_val← data_val, initializations← 10)

optim_results ← param_optimizer.optimize()
```

`optimize` gibt den Wert der Kostenfunktion auf den Trainings- und Validierungsdaten sowie die zugehörigen Parametersätze als `DataFrame` zurück. Der Nutzer kann dann entscheiden, welche Parameter er dem Modell letztendlich zuweisen möchte. Für gewöhnlich wird es sich dabei um den Parametersatz handeln, welcher mit den geringsten Kosten auf den Validierungsdaten korrespondiert. Für die Zuweisung der Parameter zu allen Teilmodelle verfügt `DynamicQualityModel` über die `SetParameters()`-Methode, siehe Algorithmus 3.

Algorithmus 3: Zuweisung der Modellparameter

```
val_min ← optim_results['loss_val'].idxmin()

QModel.SetParameters(optim_results.loc[val_min,'params_val']
```

Das dynamische Qualitätsmodell kann dann zur Prädiktion der Bauteilgüte verwendet werden, indem es simulativ mittels der `parallel_mode`-Methode auf Prozessgrößenverläufen ausgewertet wird.

Algorithmus 4: Auswertung des Modells in Bild 10 auf Prozessgrößenverläufen

```
sim_val ← QModel.parallel_mode(data_val)
```

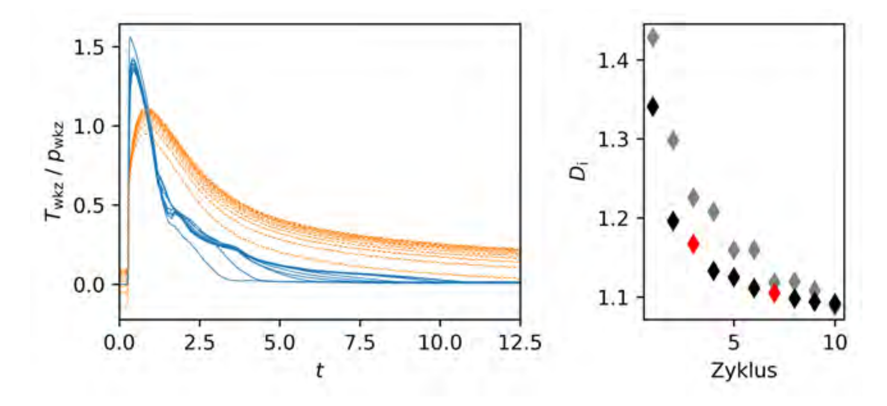


Bild 12: Links: Normierte Prozessgrößenverläufe der zehn betrachteten Produktionszyklen. Rechts: Normierter Bauteildurchmesser, Trainingsdaten in schwarz, Validierungsdaten in rot, Modellprädiktion in grau.

In Bild 12 sind die Ergebnisse der simulativen Auswertung auf den Trainings- und Validierungsdaten für einige Spritzgießzyklen mit gleichen Prozessparametern s visualisiert. Es ist deutlich zu erkennen, dass das dynamische Qualitätsmodell in der Lage ist, die variierende Bauteilqualität (bei identischen Prozessparametern s) mittels der Prozessgrößenverläufe zu prädizieren.

5 Zusammenfassung

Die (dynamische) Modellierung komplexer Produktionsprozesse ist eine herausfordernde Aufgabe. Mit der auf dem CasADi-Framework vorgestellten DIM-Toolbox wird diesbezüglich ein Beitrag zur Unterstützung von Anwendern in Industrie und Wissenschaft geleistet. Die Toolbox stellt in Klassen implementierte Modellstrukturen bereit, welche von einfachen statischen Modellen bis hin zu dynamischen schaltenden Modellen reichen. Anhand einer ausführlichen Dokumentation und bereits vorimplementierter Modelle (GRU, LSTM, MLP, Polynome) ist der Anwender in der Lage, ohne großen Aufwand eigene Modelle zu implementieren. Die Modellstrukturen sind uneingeschränkt kompatibel zu den implementierten Klassen zur Parameteroptimierung zur Lösung von Optimalsteuerungsproblemen. Alle Optimierungsverfah-

ren basieren auf dem erprobten und gut dokumentierten Löser IPOPT. Es handelt sich hierbei um ein Innere-Punkte-Verfahren, sodass auch eine Optimierung unter Nebenbedingungen möglich ist.

Neben dem Spritzgießprozess sind die Methoden der Toolbox auf andere Batch-Fertigungsprozesse übertragbar. Insbesondere solche, bei denen das Problem der Abbildung von Zeitreihen auf einen Endwert auftritt.

Danksagung

Dieses Projekt wird vom Land Hessen und dem Europäischen Fond für regionale Entwicklung (EFRE 2014-2020) gefördert, Projekt: Digital Twin of Injection Molding (DIM) FKZ: 0107/20007409.

Ein Dank an die Kollegen, insbesondere Marco Klute, des Instituts für Werkstofftechnik (IfW) - Fachgebiet Kunststofftechnik der Universität Kassel für die Planung und Durchführung der Experimente zum Zweck der Erhebung der in dieser Publikation verwendeten Daten.

Literatur

[1] J. A. E. Andersson, J. Gillis, G. Horn, J. B. Rawlings und M. Diehl. „CasADi – A software framework for nonlinear optimization and optimal control“. In: *Mathematical Programming Computation*. 2018.

[2] Y. Lu, C. Liu, I. Kevin, K. Wang, H. Huang und X. Xu. „Digital Twin-driven smart manufacturing: Connotation, reference model, applications and research issues“. In: *Robotics and Computer-Integrated Manufacturing* 61. S. 101837. 2020.

[3] P. Bibow, M. Dalibor, C. Hopmann, B. Mainz, B. Rumpe, D. Schmalzing, M. Schmitz und A. Wortmann. „Model-driven development of a digital twin for injection molding“. In: *International Conference on Advanced Information Systems Engineering*. S. 85-100. 2020.

[4] F. Tao und M. Zhang. „Digital twin shop-floor: a new shop-floor paradigm towards smart manufacturing“. In: *Ieee Access* 5. S. 20418–20427. 2017.

[5] A. Wächter und L. T. Biegler. „On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming“. In: *Mathematical programming* 106. S. 25-57. 2006.

[6] A. Rehmer, M. Klute, A. Kroll und H.-P. Heim. „An internal dynamics approach to predicting batch-end product quality in plastic injection molding using Recurrent Neural Networks“. In: *Proc. of the 6th IEEE Conference on Control Technology and Applications (CCTA).* 2022.

[7] A. Rehmer, M. Klute, A. Kroll und H.-P. Heim. „A Digital Twin for Part Quality Prediction and Control in Plastic Injection Molding“. In: *Modeling, Identification, and Control for Cyber-Physical Systems Towards Industry 4.0*, eingereicht. 2022.

KI-Ansätze bei der Entwicklung lernender Roboter – Implementierungsmöglichkeiten und Grenzen

Maja Temerinac-Ott

Hochschule Furtwangen
Robert-Gerwig-Platz 1, 78120 Furtwangen
E-Mail: maja.temerinac-ott@hs-furtwangen.de

1 Einführung

Roboter-Programme müssen für jede neue Umgebung und Aufgabe angepasst werden. Dabei müssen neue Positionen des Roboters zeitaufwendig erfasst werden bzw. neue mathematische Formeln für die Bewegungen der Roboter berechnet werden. Dabei stellt sich offensichtlich die Frage: Wie kann man Roboter einfacher programmieren?

Die Robotik gehört von Anfang an zu den aktiven Forschungsgebieten in der KI. Befeuert durch Literatur und Film entstand der Eindruck, dass Roboter immer mehr unseren Alltag bestimmen und bald schon die Menschheit beherrschen würden [1]. Tatsächlich sind wir noch weit weg davon, dass Roboter wirklich intelligent sind und in unserem Alltag die dominierende Rolle spielen. In der Industrie gibt es Roboter, die erfolgreich in der Produktion eingesetzt werden [2] , und im Haushalt haben sich Staubsauger-Roboter und Rasenmäher-Roboter etabliert. Im Bereich des autonomen Fahrens gibt es große Fortschritte [3] und auch in der Energieerzeugung, Gesundheit [4] und Ernährung [5] spielen Roboter eine immer wichtigere Rolle.

Das maschinelle Lernen hat das intelligente Verarbeiten von Sensordaten stark vereinfacht. Dabei haben sich als vorherrschende Methode künstliche Neuronale Netze (kNN) etabliert, da sie es ermöglichen einen Roboter zu program-

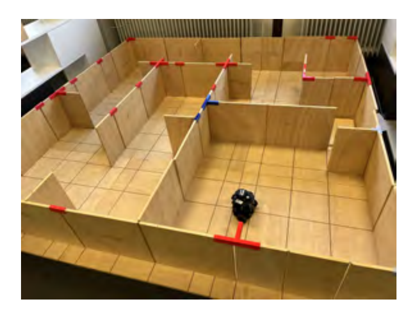
mieren, ohne dabei explizit eine Entscheidungslogik zu definieren. Ein kNN kann nur mit Hilfe von Datenbeispielen trainiert werden und imitiert somit auch das menschliche Lernen. Obwohl kNNs unglaublich gute Ergebnisse erzeugen können, scheitern sie häufig, sobald man die erlernten Algorithmen in einer leicht abgeänderten Umgebung ausprobieren möchte. Die große Herausforderung bei den kNNs ist die gelernten Modelle erklärbar zu machen und die Sicherheit der Algorithmen zu garantieren.

In diesem Beitrag werden wir auf die Programmierumgebungen (ROS [6]) und die Schnittstellen (keras/Tensorflow [7]) eingehen, die es ermöglichen Roboter mit Hilfe von maschinellem Lernen zu trainieren. Dabei werden wir insbesondere die Möglichkeiten vorstellen, wie man einen Roboter in der Simulation (gazebo) trainieren kann, um die trainierten Modelle auf echte Roboter zu übertragen. Das Testszenario besteht aus einem Holz-Labyrinth und einem Turtlebot Roboter, der mit Laser Range Scanner und einer 2D-Kamera ausgestattet ist. Dabei soll der Roboter lernen, autonom den Weg zur angegebenen Zielposition zu planen ohne dabei gegen ein Hindernis zu fahren. Es wird hierbei untersucht in wie weit die trainierten Modelle in leicht abgeänderten Szenarien funktionsfähig bleiben.

2 Maschinelles Lernen in der Simulation

2.1 Verwandte Arbeiten

Die Kombination von Deep Learning und Simulation ist vielversprechend, da sich die beiden Technologien ergänzen: Die Simulation kann sehr viele Daten erzeugen und die kNNs benötigen viele Beispiele, um sinnvolle Modelle zu lernen. In dem kNNs mit simulierten Daten trainiert werden, lernen sie komplexe Aufgaben bei der Steuerung von Robotern zu lösen [8]. Dabei kann man ohne den Roboter zu beschädigen in der Simulation viele unterschiedliche Szenarien und Parameter erforschen. Zudem bietet die Simulation die Möglichkeit mit dem Roboter zu arbeiten ohne physisch in der Anlage oder im Labor präsent zu sein. Allerdings stellt es noch immer eine große Herausforderung


(a) Turtlebot3 im Holzlabyrinth.

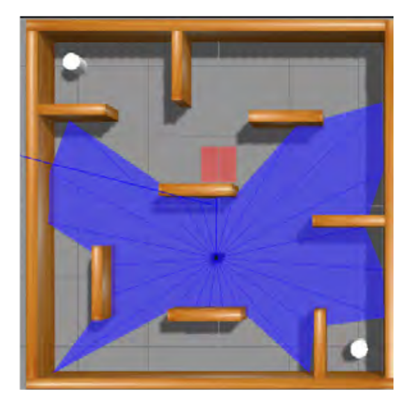

(b) Darstellung von Roboter und Simulationsumgebung in gazebo (Stage 4).

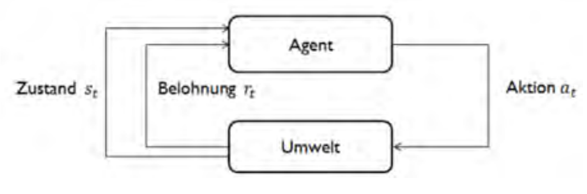
Bild 1: RL- Testumgebung für Sim2Real.

dar, die trainierten Modelle erfolgreich bei der Steuerung von echten Robotern einzusetzen [9].

2.2 ROS und Gazebo

Die Open Source Plattform ROS [6] ermöglicht es Programme für die Steuerung von Robotern zu schreiben, die sowohl auf echten Robotern als auch auf simulierten Robotern eingesetzt werden können, ohne dabei viele Änderungen im Programm durchführen zu müssen. Die Plattform basiert auf einer Master-Slave Architektur, die auf dem Publisher-Subscriber Prinzip beruht. Die Software wird in Modulen (Nodes) erstellt, die zu einem Graphen verbunden werden und über Messages miteinander kommunizieren können. Diese Messages müssen in eine Topic publiziert oder von einer Topic ausgelesen werden. ROS unterstützt verschiedenen Programmiersprachen, wobei für ML häufig die Programmiersprache Python zum Einsatz kommt.

Gazebo [10] ist eine Open Source Simulationsplattform, die leicht innerhalb von ROS installiert werden kann und seit 2004 entwickelt wird. Mittlerweile haben auch große Firmen das Potential von Simulationsplattformen erkannt und investieren in eigene Simulationsumgebungen für die Robotik i.e. [11].



(a) Modellierung der Parameter eines RL-Algorithmus.

Bild 2: RL- Kreislauf zum Erlernen von Aktionen, welche die Belohnung über die Zeit optimieren.

Gazebo stellt fertige Simulationsumgebungen zur Verfügung (z.B. in Bild 1b), erlaubt es aber auch eigene neue Simulationsumgebungen zu kreieren.

2.3 Reinforcement Learning Umgebung

Eine RL Umgebung bietet die Möglichkeit Aktionen zu erlernen, die zu einem gewünschten Ziel führen. Dabei wird die Umgebung mit Hilfe von Zuständen (s_t) modelliert, die sich über die Zeit verändern können, in dem man Aktionen (a_t) ausführt und dabei Belohnungen r_t erhält [12]. Entscheidend für die Maximierung der Belohnung in Bild 10 ist die Wahl des Agenten und die Modellierung der Umgebung. Der DQN-Agent [13] hat bei 49 Atari Computerspielen erstaunlich gute Ergebnisse erzielt und wird deswegen in dieser Arbeit eingesetzt werden. Bei der Simulation von Robotern beschränkt sich die Modellierung der Zustände auf die Modellierung der Sensordaten, wohingegen die Aktionen durch die Steuerungsbefehle, die an den Roboter gesendet werden, dargestellt werden. In Bild 1 ist der Roboter in einem Labyrinth dargestellt. Der Roboter soll mit Hilfe des RL-Algorithmus erlernen zu dem Ziel (rotes Rechteck in Bild 1b) zu gelangen ohne dabei gegen die Wand oder ein anderes Objekt zu fahren.

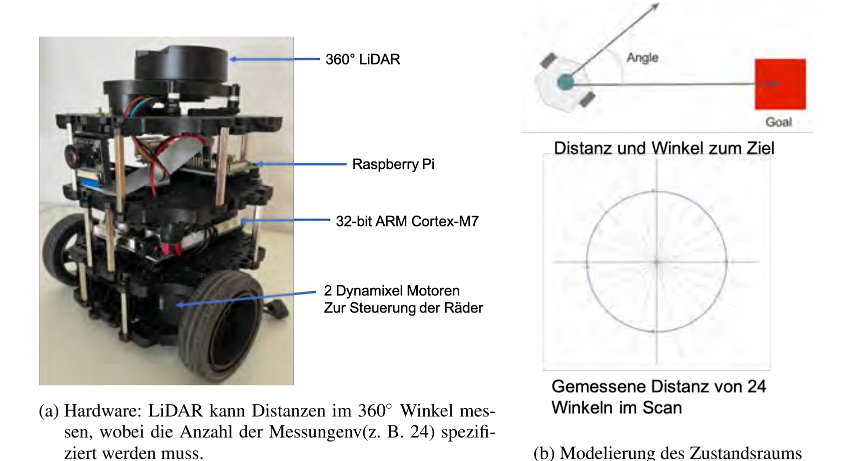
3 Experimentaufbau

3.1 Turtlebot

Der erste Turtlebot wurde 1960 am MIT entwickelt und war der Namensgeber für eine Reihe von Robotern, die mit ROS programmiert werden können und die in der Lehre und Forschung eingesetzt werden. Wir verwenden für die Experimente Turtlebot3 burger der Firma ROBOTIS [14]. In Bild 4 sieht man den Roboter, so wie die Beschriftung der wichtigsten Komponenten des Roboters. Durch den LiDar Sensor werden Distanzmessungen erzeugt und durch die Topic '/scan' innerhalb von ROS bereitgestellt und für die Modellierung des Zustands s_t verwendet. In dem man Werte in die Topic '/cmd_vel' publiziert, können die Räder angesteuert werden. Die Topic '/cmd_vel' erhält Nachrichten vom Typ *Twist*, die aus einer linearen und einer winkelbasierten Geschwindigkeit bestehen. In den Experimenten wird die lineare Geschwindigkeit konstant gehalten ($0.15 m/s$) bis das Ziel erreicht wird, während die Winkelgeschwindigkeit von dem Agenten verändert werden kann. Dabei sind folgende 5 Aktionen für a_t möglich: (1) -1.5 rad/sec, (2) -0.75 rad/sec, (3) 0 rad/sec, (4) 0.75 rad/sec und (5) 1.5 rad/sec. Durch das Verändern der Winkelgeschwindigkeit ändert der Roboter seine Bewegungsrichtung.

3.2 Installation und Modellierung des Versuchsaufbaus

Sowohl auf dem Turtlebot3 (Raspberry Pi/ ARM-Controller) als auch auf dem Steuerungs PC wurden die ROS-Pakete zu der Version ROS-kinetic installiert. Zusätzlich wurde auf dem Steuerungs-PC Tensorflow/keras installiert. Zum Trainieren des DQN-Agenten wurde die Anleitung in [14] befolgt und der DQN-Agent für Stage 1 und Stage 2 (statische Objekte) trainiert. Zu beachten ist, dass der Roboter hierbei keine Karte seiner Umgebung erstellt, sondern lediglich die Sensoreingaben benutzt, um zum Ziel zu steuern ohne dabei gegen ein Hindernis zu fahren. Die Belohnung beträgt $r_t = 200$ für das Erreichen des Ziels und $r_t = -200$ für die Kollision mit einem Hindernis. Für das kollisionsfreie Fahren wird eine Belohnung in Abhängigkeit von der Lage des Roboters zum Ziel definiert: Fährt der Roboter in Richtung des Ziels ist r_t



(a) Hardware: LiDAR kann Distanzen im 360° Winkel messen, wobei die Anzahl der Messungenv(z. B. 24) spezifiziert werden muss.

(b) Modelierung des Zustandsraums

Bild 3: Turtlebot3 burger

positiv; entfernt er sich vom Ziel wird r_t negativ. Das DQN wird trainiert, in dem der aktuelle Zustand s_t, die ausgeführte Aktion a_t und der neue Zustand s_{t+1} so wie die erhaltene Belohnung r_t an das kNN übermittelt werden. Dabei soll das kNN lernen die beste Aktion a_t für jeden Zustand s_t auszuführen. Um die beste Aktion ermitteln zu können, muss der Q-Wert $Q(s,a)$ gelernt werden, in dem sehr viele Aktionen in der Simulation ausgeführt werden.

3.3 Training des kNN

Die Architektur des verwendeten kNN ist in Fig. 4 abgebildet. Beim Deep Q-Learning werden mehrere Episoden gespielt. Jede Episode wird benutzt, um die Gewichte des Neuronalen Netzwerks anzupassen. Je mehr Episoden gespielt werden, desto besser sollte die Reward Funktion für jede Episode werden. Am Anfang sollten möglichst viele Aktionen zufällig ausgewählt werden. Mit steigender Zahl der Episoden sinkt die Anzahl der zufälligen Aktionen. Damit ältere Episoden nicht komplett vergessen werden, werden diese teilweise in der replay memory gespeichert. Diese stellt sicher, dass es keine zu abrupten Wechsel in den berechneten Parametern des Neuronalen Netzwerks

Layer (type)	Output Shape	Param #
dense_1 (Dense)	(None, 64)	1728
dense_2 (Dense)	(None, 64)	4160
dropout_1 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 5)	325
activation_1 (Activation)	(None, 5)	0

Total params: 6,213
Trainable params: 6,213
Non-trainable params: 0

Bild 4: Tensorflow-Ausgabe der kNN Archtiektur

gibt. Die Parameter des DQN sind in Tab. 1 abgebildet. Das DQN beseht aus zwei identischen kNNs, wobei die Gewichte des einen Netzwerks nach einer bestimmten Anzahl an Episoden von Source zu Target Netzwerk kopiert werden. Das Target Netzwerk trifft die Entscheidungen, während die Gewichte des Source Netzwerks mit jeder Episode neu berechnet werden. Damit das Netzwerk lernen kann, wird immer eine bestimmt Anzahl an Aktionen zufällig ausgewählt, die am Anfang noch sehr groß ist und mit der Dauer des Trainings immer kleiner wird. Bei der Aktualisierung der Gewichte im Netzwerk, wird der Einfluß der zukünftigen Aktionen durch den Parameter Discount berücksichtigt. Ein Discount nahe eins bezieht immer auch das Ende der Episode in das laufende Update der Gewichte mit ein.

4 Ergebnisse

4.1 Ergebnisse der Simulation

Die Ergebnisse zeigen, dass komplexere Umgebungen längere Trainingszeiten verlangen. In der Simulation in Bild 5 zeigen, das der Roboter nach 100 Episoden in Stage 1 (einfache Umgebung) und 1000 Episoden in Stage 2 (komplexe Umgebung) gelernt hat möglichst viele Ziele innerhalb einer Episode zu besuchen und die erhaltene Belohnungen zu maximieren. Stage 1 und Stage 2 sind Testumgebungen, welche in dem ROS-Paket ’turtlebot3_gazebo’ bereits

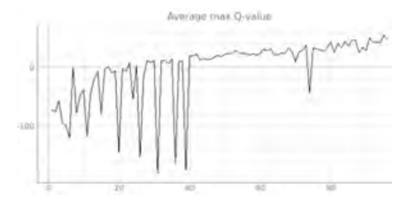
Hyperparameter	Wert
Zeitschritte pro Episode	6000
Update Rate für das Target Netzwerk	2000
Discount	0.99
Lernrate	0.00025
Startwert für die Auswahl einer Zufallsaktion	1.0
Abfallrate für die Auswahl einer Zufallsaktion	0.99
Minimaler Wert für Zufallsaktionen	0.05
Batchgröße	64
Größe der Replay Memory	1 Million (s_t, a_t)-Paare
Minimale Größe in der Replay Memory	64

Tabelle 1: Hyperparameter für das Training des DQN-Agenten.

enthalten sind. Bei Stage 1 kann der Roboter das Ziel direkt anfahren, während er bei Stage 2 lernen musste, Hindernisse zu umfahren, um zum Ziel zu gelangen, was zu einer längeren Trainingszeit führt. Bei Stage 2 wird die Kollision weniger bestraft als bei Stage 1 ($r_t = -150$). Zudem werden bei der Modellierung von s_t bei Stage 2 zwei zusätzliche Parameter betrachtet: die minimale vom Sensor gemessene Distanz und der Winkel, aus dem diese Distanz gemessen wurde.

4.2 Ergebnisse im Holzlabyrinth

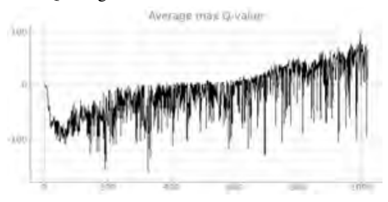
Um das trainierte Modell auf dem realen Turtlebot ausführen zu können, müssen kleinere Anpassungen getroffen werden: Die Sensordaten müssen abgetastet werden, um aus 360 Distanzmessungen des Sensors 24 Messungen für die Modellierung von s_t zu erhalten. Zudem gibtder simulierte Sensor in Gazebo den Rückgabewert *Inf* zurück, wenn der Roboter kein Hindernis detektiert, wohingegen bei dem realen Roboter der Rückgabewert Null ist. Die Angabe der Zielposition erfolgt relativ zu der Ausgangslage des Roboters. Die lineare Geschwindigkeit und die Winkelgeschwindigkeit bei dem realen Roboter entspricht der Geschwindigkeit in der Simulation.



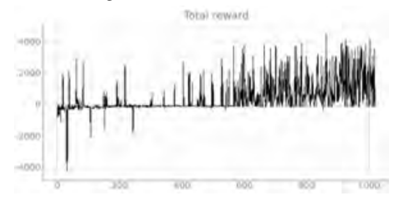
(a) Stage 1 Q-Werte während 100 Episoden des DQN-Algorithmus.



(b) Stage 1 Rewards während 100 Episoden des DQN-Algorithmus



(c) Stage 2 Q-Werte während 1000 Episoden des DQN-Algorithmus.



(d) Stage 2 Rewards während 1000 Episoden des DQN-Algorithmus

Bild 5: Training des DQN-Algorithmus

Die Simulationsumgebungen Stage 1 und Stage 2, so wie das vereinfachte Holz-Labyrinth sind in Bild 6 zu sehen. Wenn das trainierte DQN-Modell (Stage 1 oder Stage 2) auf diese neue Umgebung angewendet wird, so gelingt es dem Roboter ohne Probleme Ziele in einem Umkreis des Roboters zu erreichen, die sich auf freier Bahn befinden. Die Trefferquote entspricht der Trefferquote in der Simulation. Interessanterweise können auch Ziele erreicht werden, deren Distanz größer ist als die maximale Distanz während dem Training in der Simulation.

Soll ein Ziel wie etwa der grüne Würfel in Bild 6a erreicht werden, so bleibt der Roboter an der Abtrennung, die mit dem weißen Pfeil markiert ist hängen. Wird diese entfernt, so kann der Roboter das Ziel erreichen, allerdings nur mit dem Stage 2 DQN-Modell. Der Roboter hat dann jedoch das Problem zu wenden, wenn er wieder zurück an die Ausgangsposition fahren möchte. Hier reicht das trainierte DQN-Modell nicht aus, um alle Ziele zuverlässig zu erreichen.





(a) Einfaches Holz-Labyrinth.



(b) Stage 1



(c) Stage 2

Bild 6: Reale Testumgebung (links) für Stage 1 und Stage 2 (rechts).

5 Zusammenfassung und Ausblick

In dieser Arbeit wurde gezeigt, dass es möglich ist RL-Agenten in der Simulation zu trainieren und die trainierten Modelle auf echte Roboter zu übertragen. Sind sich Simulation und die reale Szene sehr ähnlich kann das Modell direkt übernommen werden. Um das Modell auf neue Szenarien anzupassen, könnte Active Learning hilfreich sein. Dabei sollen insbesondere Episoden zum Trainieren benutzt werden, bei denen der Mensch dem Agenten Aktionen vorgibt, um zum Ziel zu gelangen.

Weiterhin ist es interessant zu erforschen in wie weit man die trainierten Modelle von einem Roboter auf den anderen Roboter mittels Transferlearning übertragen kann. Bei dem turtlebot3 gibt es ein weiteres Modell (waffel) mit einem Greifarm, das andere Dimensionen als der turtlebot3 burger hat. Anstatt den Lernvorgang von vorne zu starten, wäre es viel effizienter die trainierten Modelle an die neuen Dimensionen anzupassen.

Schließlich ist eines der größten Risiken der kNN, das wir schlecht erklären können, was das Modell nun gelernt hat und wie zuverlässig das Modell sein

wird. Eine interessante Forschungsrichtung wäre es Methoden zu entwickeln, die das trainierte Modell erklärbar machen.

Literatur

[1] S. Russel. „Human Compatible: AI and the Problem of Control“. Viking. 2019.

[2] S. Robla-Gómez et al. „Working Together: A Review on Safe Human-Robot Collaboration in Industrial Environments“. In: *IEEE Access* 5. 2017.

[3] Y. Almalioglu, M. Turan, N. Trigoni und A. Markham„Deep learning-based robust positioning for all-weather autonomous driving“. In: *Nature Machine Intelligence* 2022.

[4] M. Kyrarini, et al. „A Survey of Robots in Healthcare“. In: *Technologies* 9 (1), 8 2021.

[5] C Rizzardo, S Katyara, M Fernandes and F Chen. „The importance and the limitations of sim2real for robotic manipulation in precision agriculture“. arXiv preprint arXiv:2008.03983. 2020.

[6] M. Quigley et al. „ROS: an open-source Robot Operating System“. In: *ICRA workshop on open source software* 3 (3.2), 5 2009.

[7] M. Abadi et al. „TensorFlow: Large-scale machine learning on heterogeneous systems“, arXiv:1603.04467 [cs.DC], 2016. Software available from tensorflow.org.

[8] J. Jesus, J. Bottega, M. Cuadros and. D. Gamarra. „Deep Deterministic Policy Gradient for Navigation of Mobile Robots in Simulated Environments“. In: *Proc., 19th International Conference on Advanced Robotics (ICAR)* 2019.

[9] S. Höfer et al. „Sim2Real in Robotics and Automation: Applications and Challenges“. In: *IEEE Transactions on Automation Science and Engineering* 18 (2). 2021.

[10] N. Koenig und A. Howard. „Design and use paradigms for Gazebo, an open-source multi-robot simulator“. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)* 2004.

[11] „NVIDIA Isaac Sim“. 2022. [Online]. Available: https://developer.nvidia.com/isaac-sim (accessed Sept. 14, 2022).

[12] S. Russell und P. Norvig. „Artificial intelligence: a modern approach“. 2002.

[13] V. Mnih et al. „Human-level control through deep reinforcement learning“. In: *Nature* 518. 2015.

[14] „ROBOTIS eManual“. [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/. [Accessed: Sept. 14, 2022].

Simulation of Synthetically Degraded Tracking Data to Benchmark MOT Metrics

Michael Hartmann[1], Katharina Löffler[1,2] and Ralf Mikut[1]

[1] Institute for Automation and Applied Informatics,
[2] Institute of Biological and Chemical Systems - Biological Information Processing,
Karlsruhe Institute of Technology
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen

Abstract

Multiple object tracking (MOT) is an essential task in computer vision, with many practical applications in surveillance, robotics, autonomous driving, and biology. To compare different MOT algorithms efficiently and select the best MOT algorithm for an application, we rely on tracking metrics that reduce the performance of a tracking algorithm to a single score.
However, there is a lack in testing the tracking metrics themselves, which can result in unnoticed biases or flaws in tracking metrics that can influence the decision of selecting the best tracking algorithm. To check tracking metrics for possible limitations or biases towards penalizing specific tracking errors, a standardized evaluation of tracking metrics is needed.
We propose benchmarking tracking metrics using synthetic, erroneous tracking results that simulate real-world tracking errors. First, we select common real-world tracking errors from the literature and describe how to emulate them. Then, we validate our approach by reproducing previously found tracking metric limitations through simulating specific tracking errors. In addition, our benchmark reveals a before unreported limitation in the tracking metric AOGM. Moreover, we make an implementation of our benchmark publicly available.

1 Introduction

Multiple object tracking (MOT) is an essential task in computer vision, with many practical applications in surveillance, robotics, autonomous driving, and biology. As MOT provides the basis for more complex tasks such as scene understanding, human-machine interaction, or analyzing cell behavior, a high tracking quality is needed. To find the most suitable tracking algorithm for an application, we rely on tracking metrics to compare different tracking approaches efficiently. Tracking metrics summarize the performance of a tracker into a single score by comparing the ground truth (GT), perfect tracking of all objects, to the tracker output and penalize any deviation from the GT, which is called a tracking error.

Tracking metrics are considered as an objective measure, thereby ignoring potential biases and limitations in the tracking metrics themselves. This unawareness towards tracking metric limitations can ultimately even lead research into the wrong direction, as the improvement of tracking approaches is quantified by the tracking measure.

Unfortunately, spotting errors in tracking metrics usually happens by chance, and handcrafted examples are generated to demonstrate them [1]. To date, limitations of several tracking metrics, such as AOGM, MOTA and IDF1 [2, 3, 4], have been reported [1, 5, 6, 7]. However, these tracking metrics are still used in benchmarks [8, 9] as developing new tracking metrics requires time. For instance, in 2013, Leichter and Krupka published several problems of the popular MOTA metric [6]. Eight years later, the HOTA metrics was proposed, which aims to replace MOTA by claiming to be more balanced [5].

A systematic approach to evaluate tracking metrics is needed to ensure that tracking metrics align with the established objectives that tracking algorithms should meet, and to facilitate the development of tracking metrics themselves. For example, one concept which is important for the targeted improvement of tracking metrics is error differentiability [6] – the separate quantification of the tracking performance concerning different types of tracking errors. Until now, most metrics only provide a single composite score, which does not indicate how the metric penalizes different types of errors. By providing an approach

that allows to systematically analyze the behavior of the metric, different types of errors, biases and limitations of metrics towards specific error types can be detected.

Moreover, such an approach can help to boost acceptance of a new tracking measure in the community as the consistency of the proposed metric concerning desired properties can be demonstrated. For instance, a desirable property of tracking metrics is monotonicity – a metric score should improve if, for example, a tracking error is removed from the dataset [6].

We propose benchmarking tracking metrics by replacing the tracking algorithm with synthetic tracking results emulating real-world tracking errors. We select a set of frequently occurring real-world tracking errors and provide instructions on how to simulate them. To validate our approach, we reproduce already reported tracking metric limitations by simulating specific tracking errors. Our benchmark reveals a before unreported limitation of the commonly used tracking metric AOGM. In addition, we make an implementation of our benchmark publicly available at: *https://github.com/mrhartmann/benchmark-mot-metrics*. This work presents the method and main results of the Bachelor's thesis by Hartmann [10].

The concept of creating synthetically degraded tracking data for evaluation is established: For instance, Schott synthetically degraded tracks to investigate the robustness of extracted features to describe tracks [11], whereas Löffler et al. synthetically degraded segmentation data to investigate the robustness of tracking algorithms when provided with erroneous segmentation data [12]. However, to the best of our knowledge, we are the first proposing to synthetically degrade tracking data to evaluate tracking metrics.

The remainder of this paper is organized as follows: First, we describe how erroneous tracking data can be used to investigate tracking metrics and introduce how common tracking errors can be emulated. Then, we generate a benchmark data set of erroneous tracking results to evaluate popular MOT metrics. Finally, we discuss our findings and the limitations of our approach.

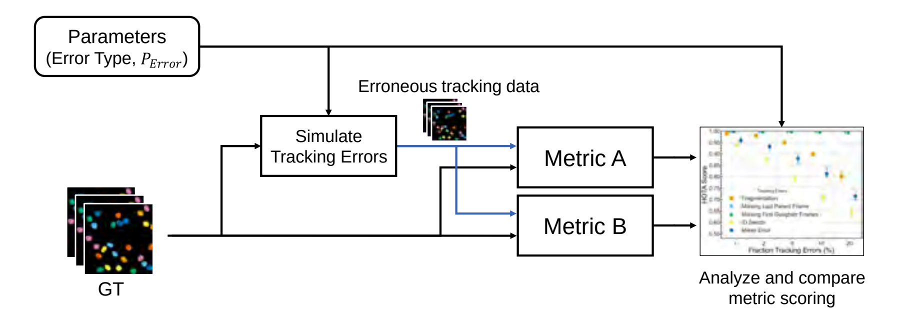


Figure 1: We degrade perfect GT data with simulated tracking errors to create erroneous tracking results, which are evaluated together with the GT by the metrics. The metric can be assessed by comparing the input parameters – the selected tracking errors and their percentage P_{Error} in the data set – with the resulting metric scoring.

2 Methods

We propose to degrade perfect GT data with simulated tracking errors to create erroneous tracking data. The synthetically degraded tracking data and perfect GT are forwarded to the tracking metric for evaluation. The flow of the data is shown in Figure 1. With the simulation of tracking errors, metric development becomes independent of the tracking algorithms that are commonly used to produce the tracking results needed for evaluation. To investigate whether a metric fulfills the property of monotonicity, the fraction of tracking errors can be chosen arbitrarily. We emulate the real-world tracking errors: ID switches, fragmentation, and mitosis errors which can be emulated separately or together to create degraded data covering several types of tracking errors. As GT we assume that each track in the GT is given by its segmentation masks, where segmentation masks belonging to the same track have the same ID. In addition, to model mitosis errors, a lineage file which indicates predecessor-successor links is needed.

In the following, we introduce the selected types of tracking errors by describing where they occur in real-world tracking scenarios and how we simulate them.

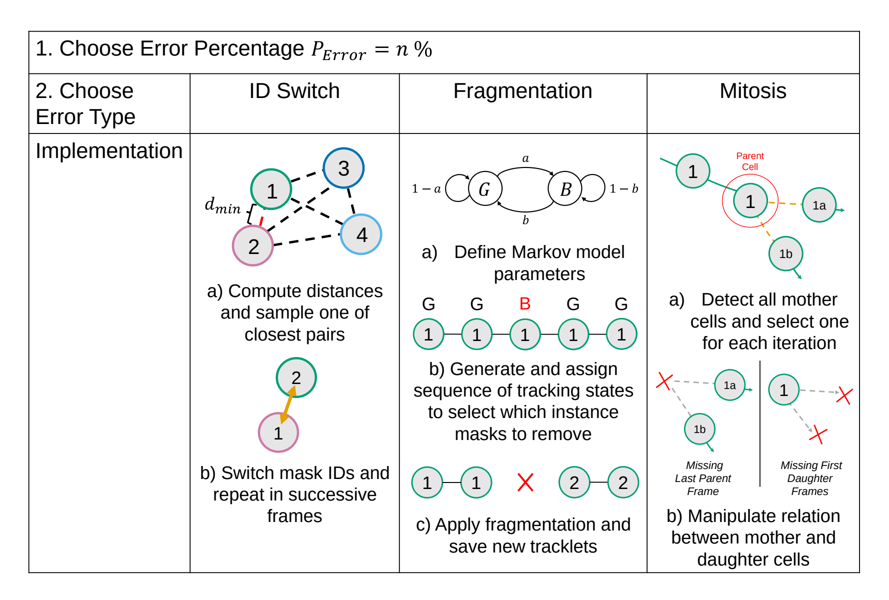


Figure 2: Simulation of tracking errors. An error percentage and tracking error type is selected by the user to degrade the perfect GT data. The implementation is visualized for each error type. The circles are instance masks which are connected by links to form tracks. The processing steps are reiterated until the desired error percentage is reached in the data set.

2.1 ID Switches

Fast movements of objects can result in two objects switching positions between frames, causing ID switch tracking errors. Additional sources of ID switches are unpredictable changes of direction, erroneous detection of objects, or wrongly classified mitosis events [13, pp. 2124-2125]. A missing detection creates a scenario, where the tracking algorithm associates another mask to the ID, if it is close to the position of the original mask in the previous frame, causing a switch in tracks that propagates through the following time steps. This error exists even in data with perfect detection, as it is also reliant on the distance measure and linking method of the tracking algorithm. This error is handled separately from fragmentation in metrics [2, 3, 4] since preserving the ID is vital for many MOT fields [2, 3, 4, 5].

Simulation

We simulate ID switches by swapping the IDs of close tracks. Therefore, the Euclidean distance between all pairs of segmentation masks is calculated in each frame. The pair sampling approach is adapted from the merging operation for segmentation masks from Löffler et al. [12]. Based on the Euclidean distances between segmentation masks, we identify for each track its closest neighbor track. Then, for each ID switch, we sample a pair of tracks from the remaining 100 closest pairs of tracks, where each pair of tracks is assigned a sampling weight inversely proportional to their minimum distance. Hence, tracks which are closer are likelier to be sampled for an ID switch. This process, as visualized in Figure 2, is repeated until the desired fraction of ID switches in the dataset is reached. In addition, for cell data sets the daughter tracks need to be assigned to the switched ID to keep the predecessor-successor information intact.

2.2 Fragmentation

Fragmentation results from missing detections, which are often referred to as False Negatives (FN). Missing detections can originate from illumination variation, fluorescent marker wear off, shadows and occlusion, and more [9, p. 22]. As this error occurs frequently, many common performance metrics include a FN tracking error [2, 3, 4, 5]. Moreover, fragmentation can lead to additional tracking errors, for example ID switches and mitosis division ambiguities [11, p. 32].

Simulation

To fragment tracks, a two-state Markov model is used. Like the Markov model introduced by Schott [11], one state represents good tracking quality (G), whereas the other state represents bad tracking quality (B). We use this Markov model to generate fragmented tracks, by generating a sequence of states of the same length as a track, and assigning each instance mask at position k in the track the state at position k of the state sequence. Instance masks that are assigned

to the good tracking quality state G will remain, whereas instance masks that are assigned the bad tracking quality state B will be removed. This process is visualized in Figure 2.

To achieve the desired fragmentation – the fraction of deleted instance masks and the length of the resulting gaps – the transition probabilities a and b between the states can be adjusted. Instead of specifying a and b directly, we propose how transition probabilities can be calculated from two intuitive user input parameters: the desired error percentage (P_{Error}) and the fragmentation gap length (l_{Gap}).

The probability of returning to the good tracking state G in n steps, is $b(1-b)^{n-1}$, as the model stays $n-1$ steps in the bad tracking state B which has probability of $1-b$, before transitioning to G with probability b. Hence, the expected time until the model returns to the good tracking state $E[T_G]$, can be rewritten as a simple fraction by using the geometric sum

$$E[T_G] = \sum_{n=1}^{\infty} nb\,(1-b)^{n-1} = \frac{1}{b}, \quad \text{for } |1-b| < 1\;. \tag{1}$$

While $E[T_G]$ is the expected time until the model returns to the G state, it can also be interpreted as the average time spent in the B state. As instance masks which are assigned the B state will be removed, which creates gaps in the track, the average time spent in the B state can also be referred to as the gap length l_{Gap}. Hence, the transition probability b can be computed by specifying the user input parameter l_{Gap}

$$b = \frac{1}{E[T_G]} = \frac{1}{l_{\text{Gap}}}\;. \tag{2}$$

From b the missing parameter a can be computed using the steady state theorem [14, p. 176]

$$\pi_{\text{eq}} = \left(\frac{b}{a+b}, \quad \frac{a}{a+b}\right)\;. \tag{3}$$

The probabilities of the steady state can be set using the desired error percentage P_{Error}, which is specified by the user. As instance masks are only removed in the bad tracking state, and kept in the good tracking state we set π_{eq}

$$\pi_{\text{eq}} = \left(1 - P_{\text{Error}}, \quad P_{\text{Error}}\right) , \tag{4}$$

to reach the desired fraction of fragmentation errors. The transition probability for a can be calculated by combining Equation 3 and Equation 4

$$a = \frac{P_{\text{Error}} b}{1 - P_{\text{Error}}} . \tag{5}$$

The fragmentation is applied iteratively until the desired fraction of fragmentation errors is reached. To simulate different gap lengths, the fragmentation gap length parameter l_{Gap} can be adjusted. If no l_{Gap} is provided, the steady state of a Markov model π_{eq} is used, as given a long enough time, it provides an approximation of how long the Markov model will stay in each state. We set the probability a – switching to the bad tracking quality state B – equal to P_{Error}, whereas b – switching to the good tracking quality state G – is set to $1 - P_{\text{Error}}$, resulting in

$$b = \pi_{\text{eq},1} = 1 - P_{\text{Error}}, \quad a = \pi_{\text{eq},2} = P_{\text{Error}} . \tag{6}$$

2.3 Mitosis Tracking Errors

In cell data, an additional type of tracking error exists, which can occur due to missing detections or False Positives. During mitosis, cells face large changes in their scale and shape. As the temporal resolution of cell data sets is usually very low, the changes between two successive frames can be substantial and therefore can lead to erroneous detections or links [13, p. 1]. Moreover, the simultaneous division of nearby cells can lead to a high density of cells, further complicating the correct association between predecessor and successor tracks.

Simulation

We base the simulation of the mitosis tracking errors on the showcases first described by Chen et al.: Single Daughter Frame Missing, Last Mother Frame Missing and Both Daughter Frames Missing [1]. In addition, we added the No Mitosis Detection and Single Daughter Link Detected cases, which can be modeled by manipulating the lineage file only. To degrade mother-daughter links, first all mother cells are extracted using the lineage information from the GT. The mother-daughter links are then altered by removing the link from the lineage information or also adding fragmentation errors to the mother and daughter tracks. This process is visualized in Figure 2.

3 Experiment

To demonstrate our approach, we generate synthetically degraded tracking data using the just introduced tracking errors and evaluate four MOT metrics on them.

3.1 Data

To create synthetically degraded data sets, we select microscopy data showing cells as these data comprise all challenges encountered in general MOT and in addition contain splitting objects (cell divisions). We convert the synthetically degraded tracking data into two different file formats for metric evaluation, to evaluate specialized cell tracking metrics and general MOT metrics.

The synthetically degraded tracking data is generated by modifying the ground truth masks of the Fluo-N2DH-SIM+ 02 data set, from the Cell Tracking Challenge (CTC) [9]. We generate fractions of $n = 1, 2, 5, 10, 20\,\%$ of errors for each tracking error type and create for each combination of tracking error type and error fraction $N = 10$ runs. Every synthetically created tracking dataset is evaluated on the tracking metrics and the tracking score is averaged over the ten runs for each combination of tracking error type and error fraction.

3.2 Metrics

For evaluation, we select the metrics MOTA [3], IDF1 [4], HOTA [5], and TRA [2]. All metrics range between 0 and 1, where a higher score refers to a better tracking result. MOTA and IDF1 have been the most popular general MOT metrics and are widely used in benchmarks such as the MOTChallenge [8]. The recently proposed HOTA metric has been claimed to resolve issues of IDF1 and MOTA [5]. TRA is a normalized version of the AOGM [2] metric, which is used in the Cell Tracking Challenge benchmark [9]. Some flaws of TRA were already reported by Chen et al. [1].

3.3 Results

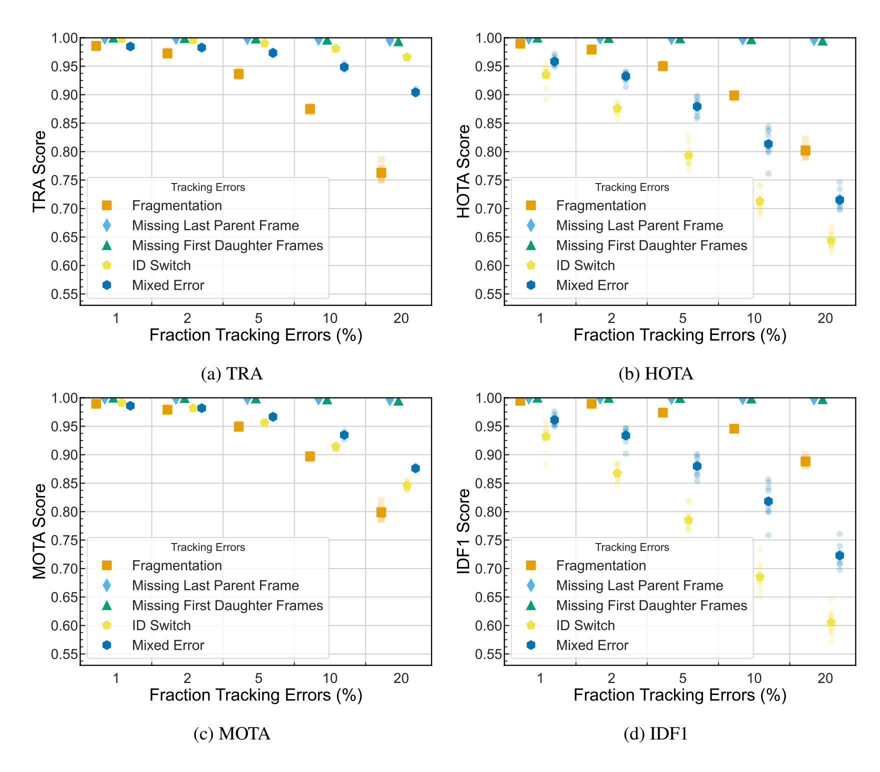
In the following, we analyze the impact of different tracking errors on the selected MOT metrics.

Effect of Different Errors on the four Metrics

First, the effect of different tracking errors – fragmentation, ID switches, mitosis errors, and a mixed error – on the metric scores of TRA, HOTA, MOTA, and IDF1 is analyzed, which is shown in Figure 3. For the mixed errors, the error percentage is split equally between the three tracking error types: $\frac{1}{3}$ of fragmentation, $\frac{1}{3}$ of ID switches, and $\frac{1}{3}$ of mitosis missing daughter frames errors.

For all tracking errors, a higher error percentage leads to a reduced metric score. For all metrics, mitosis errors are penalized the least. For IDF1 and HOTA, in Figure 3, the ID switch error has the biggest impact on the final metric scores. In contrast to MOTA and TRA, the same ID switch data sets are scored considerably lower by IDF1 and HOTA, with a score as low as 0.6 for IDF1. This is accompanied by notable differences in the metric scores between each error type for IDF1 and HOTA.

The cell tracking metric TRA, in Figure 3a, is effected strongly by the fragmentation, whereas ID switches have a low impact on the score. The TRA



(a) TRA

(b) HOTA

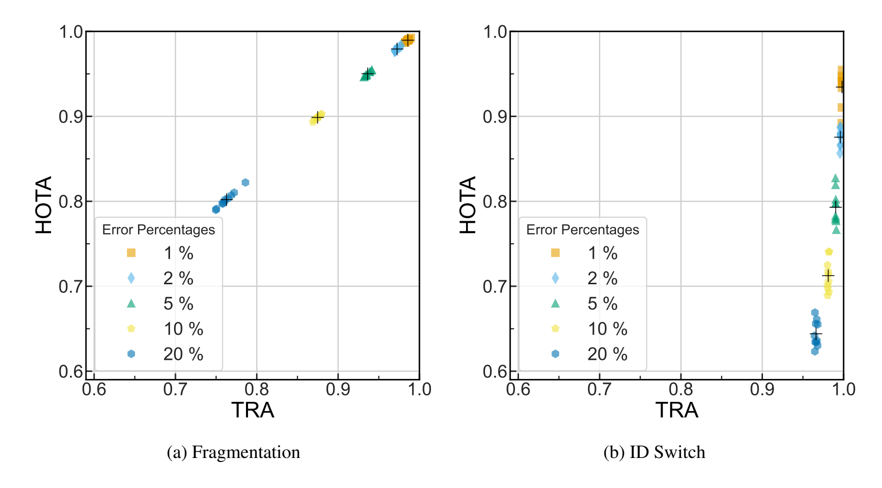
(c) MOTA

(d) IDF1

Figure 3: Evaluation of erroneous tracking data sets on the metrics. The link to the predecessor is set for all cases. For each run, a fixed percentage of ground truth tracks is modified to simulate tracking errors. The $N = 10$ single runs are shown in translucent markers, whereas the average of all runs is shown in solid color.

metric scores ID switches and mixed errors similarly and penalizes these errors only slightly, whereas the fragmentation scoring declines rapidly for increasing fractions of tracking errors.

For MOTA, Figure 3c shows a similar decline in score for ID switches, fragmentation and mixed errors.



(a) Fragmentation

(b) ID Switch

Figure 4: Comparison of the effect of fragmentation and ID switches on the TRA and HOTA metric. For each run, a fixed percentage of ground truth tracks is modified to generate tracking errors. The $N = 10$ single runs are shown as circles, whereas the average of all runs is shown as a black cross.

Comparison of Metric Scores for Fragmentation and ID Switches

Next, the HOTA and TRA metric are compared concerning their scoring of the tracking errors fragmentation and ID switches, which is shown in Figure 4. Both metrics show a similar decline when fragmentation errors increase. In contrast, ID switches affect HOTA stronger – 1 % of ID switch errors result in a score under 0.95 – which is followed by a steep drop in the metric score for higher percentages. The effect of ID switches on the TRA score is considerably lower – 20 % of ID switch errors result in a score larger than 0.95.

Difference of Keeping or Ignoring Predecessor Information for TRA Metric

Chen et al. first spotted an issue with the TRA measure concerning mother-daughter links around mitosis events with FN errors [1] based on small show-case examples. Using the different implemented mitosis errors, we reproduce this error scenario to investigate how much this limitation influences the final

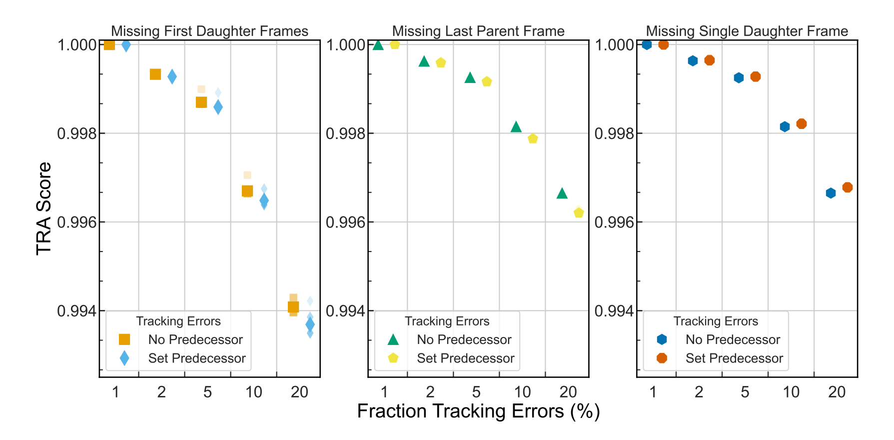


Figure 5: Difference of Linking the Predecessor for the Mitosis evaluation on TRA. For each run, a fixed percentage of ground truth tracks is modified to simulate tracking errors. The predecessor ID is either set or removed for modified tracks. The $N = 10$ single runs are shown in translucent markers. The average of all runs is shown in solid color.

metric score in a larger dataset. Each of the five plots in Figure 5 and Figure 6 includes two separate evaluations of synthetically degraded tracking data, one with and one without keeping the predecessor information for the erroneous tracks. Starting with two identical data sets, while applying tracking errors, the predecessor of the manipulated track is kept in the first and removed in the second.

A comparison for mitosis, fragmentation, and mixed errors is done, to analyze whether the TRA metric wrongly penalizes this correct information of the predecessor.

Overall, mitosis tracking errors, as shown in Figure 5, have a small effect on the TRA metric score. There is also nearly no variation of the scores between runs. The first two plots in Figure 5 score mitosis cases where either the last frame of the predecessor track is missing, or the first frames of both successor tracks are missing. In both cases, keeping the predecessor information results in worse scores by the TRA metric. The last plot in Figure 5 shows the case of a single missing successor frame, which is scored higher for keeping the predecessor information. The influence of the different mitosis errors on the TRA score decreases from left to right in Figure 5.

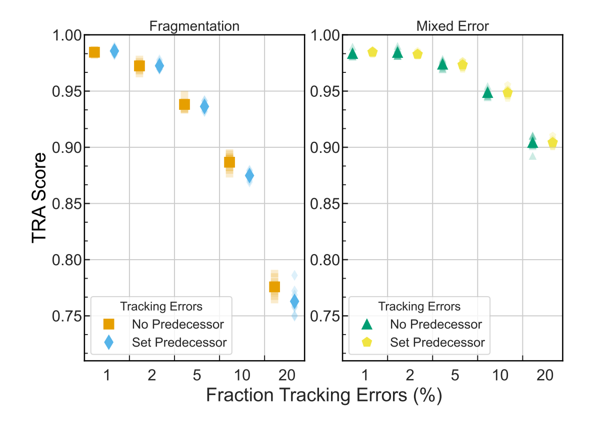


Figure 6: Difference of Linking the Predecessor for fragmentation and mixed tracking errors on the TRA metric. For each run, a fixed percentage of ground truth tracks is modified to simulate tracking errors. The predecessor ID is either set or removed for modified tracks. The $N = 10$ single runs are shown in translucent markers. The average of all runs is shown in solid color.

Figure 6 shows the influence of keeping the predecessor information on the TRA score in case of fragmentation and mixed errors. The fragmentation tracking errors in the left plot of Figure 6, are scored lower for setting the predecessor information, for error percentages of $5-20\,\%$.

4 Discussion

As mentioned by Luiten et al., IDF1 and MOTA have a bias towards detection and association, respectively [5]. Using the proposed benchmark, we could reproduce this observation as shown in Figure 3. The IDF1 score is strongly effected by ID switches, whereas fragmentation has a very low impact, as shown in Figure 3d. For MOTA, the mentioned bias towards detection is also visible in Figure 3c, as fragmentation is penalized the most.

Concerning the proposed alternative HOTA [5], ID switches and mixed errors are penalized similarly to IDF1, whereas fragmentation is penalized similarly to MOTA. These initial observations suggest that HOTA has a better balance

between detection and association, in contrast to MOTA or IDF1 alone, although the influence of ID switches on the HOTA score is still very strong.

Association errors have a very low impact on the TRA score, whereas fragmentation errors have a large impact, as shown in Figure 3a. The strong bias towards detection in the TRA metric is caused by high penalties for FN errors. The experiments with different types of mitosis errors, shown in Figure 5, support the statement of [1] that for AOGM and hence for TRA, which is derived from AOGM, it is more advantageous to ignore the information about the predecessor than to keep it in certain cases of mitosis errors. Although TRA is developed for cell tracking, erroneous mitosis detection is penalized very little, although the lineage of a cell is of high interest for instance during embryonic development [15, 16]. Additional metrics should be taken in consideration when comparing the quality of different cell tracking algorithms.

Moreover, using the proposed benchmark we could discover a yet unreported limitation of the AOGM and hence the TRA metric. If a track is fragmented, storing this information, for evaluation with the TRA metric, requires creating several tracks and link each track to its previous track. However, keeping this information is scored worse than discarding this information by not linking to the previous track. This observation matches with the already mentioned, flawed scoring of mitosis errors. Taken together, these observations reveal a weakness of the file format used in the TRA metric: the parent ID column indicates fragments belonging to the same track as well as mother-daughter relationships after cell division.

Limitations

Although we emulated real-world tracking errors, the simulated tracking results can in some cases appear artificial – e.g. long gaps (fragmentation) but the predecessor link is kept. Comparing the impact of tracking errors on different metrics should be done cautiously, especially when comparing different metrics which each other, as the method by which the error percentage is achieved differs. For fragmentation, each track has on average n% of its segmentation masks removed. For mitosis errors, only n% of the mitosis events

are modified. For ID switches, n% of tracks are selected for which their ID is switched pairwise. The results were computed with a limited amount of $N = 10$ runs for each combination of tracking error type and fraction of tracking errors. A larger number of runs is required to examine the variation between the runs in more detail. Moreover, we applied our method just to data from the microscopy image domain, so the method should be applied to data from other domains as well. Also, different tracking metrics can require a different storing of track and lineage information. When comparing metrics, the file format might not always include the same information and thus result in an unfair comparison.

5 Conclusion

We propose benchmarking tracking metrics by replacing the tracking algorithm with a method to synthetically degrade tracking data, emulating a set of frequently occurring real-world tracking errors. In a first analysis, we reproduced already reported limitations of popular tracking metrics and discovered a limitation of the AOGM metric.

Directions of future work are the extension of the proposed concept to provide a standardized approach to evaluate tracking metrics, using the approach to investigate tracking metrics that rank tracking algorithms without requiring a ground truth, and the adaption of the proposed idea to develop benchmarks for metrics used in other tasks than MOT.

References

[1] Ye Chen and Yuankai Huo. “Limitation of acyclic oriented graphs matching as cell tracking accuracy measure when evaluating mitosis”. In: *Medical Imaging 2021: Image Perception, Observer Performance, and Technology Assessment*, pp. 30–35, 2021.

[2] Pavel Matula, Martin Maška, Dmitry V. Sorokin, Petr Matula, Carlos Ortiz-de-Solórzano, and Michal Kozubek. “Cell tracking accuracy

measurement based on comparison of acyclic oriented graphs". In: *PLOS ONE* 10(12): 1–19, 2015.

[3] Keni Bernardin and Rainer Stiefelhagen. "Evaluating multiple object tracking performance: the CLEAR MOT metrics". In: *EURASIP Journal on Image and Video Processing* 2008(1): 1–10, 2008.

[4] Ergys Ristani, Francesco Solera, Roger S. Zou, Rita Cucchiara, and Carlo Tomasi. "Performance measures and a data set for multi-target, multi-camera tracking". arXiv: 1609.01775. Sept. 6, 2016.

[5] Jonathon Luiten, Aljos A. Os Ep, Patrick Dendorfer, Philip Torr, Andreas Geiger, Laura Leal-Taixé, and Bastian Leibe. "HOTA: a higher order metric for evaluating multi-object tracking". In: *International Journal of Computer Vision* 129(2): 548–578, 2021.

[6] Ido Leichter and Eyal Krupka. "Monotonicity and error type differentiability in performance measures for target detection and tracking in video". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35(10): 2553–2560, 2013.

[7] Martin Maška, Vladimír Ulman, David Svoboda, Pavel Matula, Petr Matula, Cristina Ederra, Ainhoa Urbiola, Tomás España, Subramanian Venkatesan, Deepak M.W. Balak, et al. "A benchmark for comparison of cell tracking algorithms". In: *Bioinformatics* 30(11): 1609–1617, 2014.

[8] Patrick Dendorfer, Aljoša Ošep, Anton Milan, Konrad Schindler, Daniel Cremers, Ian Reid, Stefan Roth, and Laura Leal-Taixé. "MOTChallenge: a benchmark for single-camera multiple target tracking". In: *International Journal of Computer Vision* 129(4): 845–881, 2021.

[9] Vladimír Ulman, Martin Maška, Klas E. G. Magnusson, Olaf Ronneberger, Carsten Haubold, Nathalie Harder, Pavel Matula, Petr Matula, David Svoboda, Miroslav Radojevic, et al. "An objective comparison of cell-tracking algorithms". In: *Nature Methods* 14(12): 1141–1152, 2017.

[10] Michael Hartmann. "Simulation of Synthetically Degraded Tracking Data to Benchmark Multi Object Tracking Metrics". Bachelor's thesis. Karlsruhe: Karlsruhe Institute of Technology (KIT), 2022.

[11] Benjamin Schott. "Interactive and Quantitative Knowledge-Discovery in Large-Scale 3D Tracking Data". PhD thesis. Karlsruhe: Karlsruhe Institute of Technology (KIT), 2018.

[12] Katharina Löffler, Tim Scherr, and Ralf Mikut. "A graph-based cell tracking algorithm with few manually tunable parameters and automated segmentation error correction". In: *PLOS ONE* 16(9): 1–28, 2021.

[13] Seungil Huh, Sungeun Eom, Ryoma Bise, Zhaozheng Yin, and Takeo Kanade. "Mitosis detection for stem cell tracking in phase-contrast microscopy images". In: *2011 IEEE International Symposium on Biomedical Imaging: from Nano to Macro (ISBI 2011)*, pp. 2121–2127, 2011.

[14] Robert G. Gallager. "Discrete Stochastic Processes". Vol. 321. The Springer International Series in Engineering and Computer Science, Communications and Information Theory, 1996.

[15] Khaled Khairy and Philipp J. Keller. "Reconstructing embryonic development". In: *genesis* 49(7): 488–513, 2011.

[16] Andrei Y. Kobitski, Jens C. Otte, Masanari Takamiya, Benjamin Schäfer, Jonas Mertes, Johannes Stegmaier, Sepand Rastegar, Francesca Rindone, Volker Hartmann, Rainer Stotzka, et al. "An ensemble-averaged, cell density-based digital model of zebrafish embryo development derived from light-sheet microscopy data with single-cell resolution". In: *Scientific Reports* 5(1): 8601, 2015.

OResNet: Ein flexibles ResNet für Octrees

Stefan Schütte, Torsten Bertram

Lehrstuhl für Regelungssystemtechnik, Technische Universität Dortmund
Otto-Hahn-Str. 8, 44227 Dortmund
E-Mail: {stefan.schuette,torsten.bertram}@tu-dortmund.de

1 Einführung

Die lernbasierte Perzeption einer dreidimensionalen Umgebung wird nach wie vor von einer oft speicherintensiven Datenrepräsentation gehemmt. Dies beschränkt die Anwendung von Methoden des maschinellen Lernens, wie Convolutional Neural Networks (CNNs) im dreidimensionalen Raum auf kleine Anwendungsräume, geringe räumliche Auflösungen oder leistungsfähige Rechenhardware. Für übliche dreidimensionale Daten werden bei dicht besetzten Voxel-Repräsentationen große Speicherbereiche für nicht belegte Voxel aufgewendet. Alternativ können Punktwolkendaten direkt vom Netz verarbeitet werden. Hierbei berücksichtigt jedoch die Netzstruktur den räumlichen Zusammenhang der Eingabedaten nicht mehr explizit. Spärlich besetzte dreidimensionale Tensoren werden zunehmend für die Aufgabe der räumlichen Segmentierung eingesetzt. In diesen Strukturen werden Operatoren wie die aus CNNs bekannte Faltung nur auf besetzte Voxel angewendet, während bei allen unbesetzten Voxeln angenommen wird, dass alle Merkmale 0 sind. Dieses Vorgehen reduziert die Anzahl an notwendigen Operationen sowie den Speicherbedarf der Datenstruktur. Die große räumliche Ausdehnung des resultierenden Tensors beschränkt das *Receptive Field* des Netzes für eine gegebene Tiefe und Kernelgröße. Die Struktur eines Octrees kann für die weitere Datenverarbeitung nützlich sein, da Operationen wie Kollisionsprüfungen in Octrees effizient durchgeführt werden können. In der Robotik wird das OctoMap-Format [4] für verschiedene Aufgaben in statischen dreidimensionalen Umgebungen verwendet. Ein neuronales Netz, das die Belegung und semantische Klassen einzelner

Voxel in einem Octree bestimmt, kann somit als Vorverarbeitungsschritt für die Kartenerstellung gesehen werden. Die zentrale Idee dieses Beitrags ist daher, ein flexibles Netz für die Segmentierung dreidimensionaler Daten zu entwickeln, das die räumliche Struktur der Eingangsdaten nutzen kann.

2 Verwandte Arbeiten

Für Bildsegmentierung werden häufig Architekturen wie U-Net [9] verwendet, da sie Bilder effizient pixelweise segmentieren können, indem skalierte Merkmalskarten als Zwischenrepräsentationen genutzt und fein aufgelöste Merkmale über *Skip Connections* auf die Ausgangsseite übertragen werden. Durch die begrenzte Größe üblicher Bilddaten können diese Operationen im zweidimensionalen Raum auf dicht besetzten Zwischenrepräsentationen durchgeführt werden. Dies lässt sich nicht direkt auf den dreidimensionalen Raum übertragen. Hier muss entweder die Auflösung begrenzt oder eine spärlich besetzte Darstellungsform gewählt werden, um den Speicherbedarf zu reduzieren. Beispiele für solche spärlich besetzten Repräsentationen bilden Methoden wie PointNet [8] und davon abgeleitete Architekturen [15]. Alternativ können dreidimensionale CNNs auf spärlich besetzte Tensoren angewendet werden, die die Eingabedaten in einer strukturierten Form darstellen. Dieses Vorgehen wird von Ansätzen wie Minkowski U-Net [2] oder SPVNAS [11] verwendet. Beide Ansätze bauen im Netz eine hierarchische Struktur auf, die jedoch für die Ein- und Ausgabe nicht verwendet wird. Methoden wie O-CNN [14] nutzen dagegen explizit eine hierarchische Struktur, den namensgebenden Octree, sowohl in der Eingabe als auch der Ausgabe. Die Eingabe wird auch hier durch eine U-Net-Struktur geführt. Eine entscheidende Einschränkung ist die erzwungene Verwendung der Eingabestruktur für die Ausgabe. Dies kann bei einer Segmentierung zu einem erhöhten Speicherbedarf führen, wenn größere Bereiche in der Ausgabe zur selben Klasse gehören. Weiterhin müssen auch alle Berechnungen bis zum Knoten $\mathbf{c}_{d_{\max},x,y,z}$ auf der untersten Ebene durchgeführt werden, egal ob eine frühzeitigere Klassifikation bereits möglich wäre. So werden mehr Berechnungen als zwingend notwendig durchgeführt. Daher bietet es sich an, eine Struktur wie O-CNN [14] zu erweitern, sodass eine flexible Prädiktion der Ausgabestruktur ermöglicht wird. Dieser Beitrag

untersucht die Güte eines kombinierten Netzes zur Segmentierung und Formrekonstruktion für eine Punktwolken-Segmentierungsaufgabe.

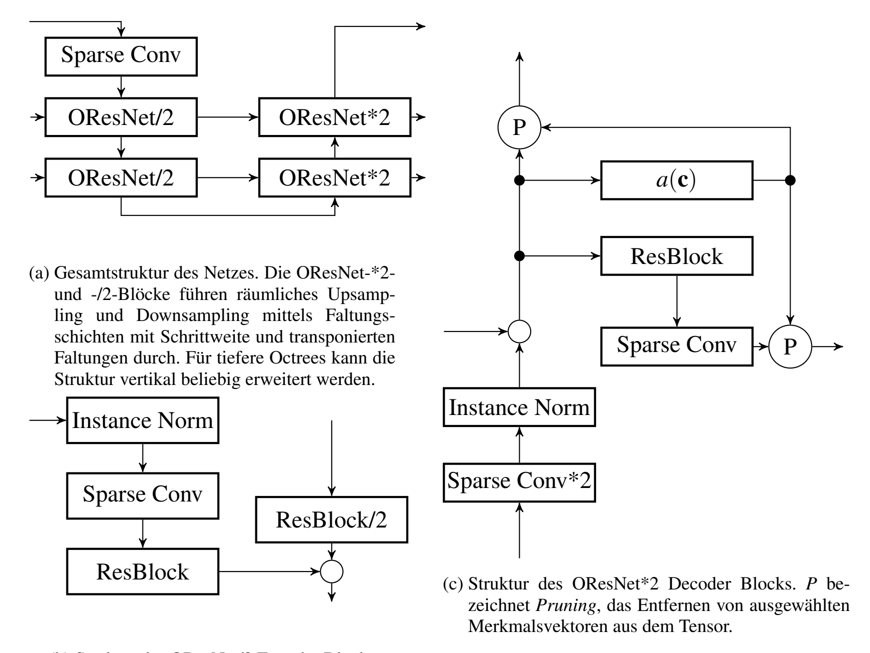
3 Ein flexibles ResNet für Octrees

Residual Neural Networks (ResNets) [29] werden für unterschiedliche Aufgaben in der Bildverarbeitung eingesetzt. Neuronale Netze werden auch in der Literatur auf Octrees angewendet [14]. Die Ausgabestruktur ist dabei jedoch häufig durch die Eingabedaten vorgegeben. Dadurch können Probleme bei der Definition von Kostentermen umgangen werden, die Flexibilität des Netzes, beliebige Strukturen in der Ausgabe darzustellen geht jedoch verloren. Im Gegensatz zu [14] wird bei dem in dem vorliegenden Beitrag beschriebenen Modell keine gemeinsame räumliche Struktur von Ein- und Ausgabe erzwungen. Stattdessen wird die Struktur, inspiriert von [12], gemeinsam mit der semantischen Information aus den Trainingsdaten gelernt und generalisiert. Das zur gleichzeitigen Inferenz von Struktur und Semantik verwendete Modell wird in Abbildung 1a dargestellt.

Zusätzlich zur Klassenausgabe für jede aktive Zelle auf jeder Ebene prädiziert der OResNet *2-Block eine Aktivitätsfunktion $a(\mathbf{c})$, die die Belegung der Zelle $\mathbf{c}$ beschreibt. $a(\mathbf{c})]$ besteht aus einem ResBlock und einer Sparse-Conv-Schicht, die jeweils einen 1×1-Kernel besitzen, um die räumliche Struktur des Tensors $\mathbf{C}$ nicht zu verändern.

3.1 Lernen von Segmentierung in hierarchischen Strukturen

Da die hierarchische Struktur von Octrees sich mit dem Besetzungszustand von Zellen auf höheren Ebenen ändern kann, stellt das Lernen einer Segmentierung auf derartigen Daten ein Problem für die Formulierung üblicher Kostenterme dar. Sollte das Netz eine Zelle als belegt bewerten, die in der Ground Truth nicht als belegt festgelegt wird, kann die Cross Entropy hierfür keine Kosten bestimmen und damit auch keinen Gradienten, der den Fehler des Netzes in



(a) Gesamtstruktur des Netzes. Die OResNet-*2- und -/2-Blöcke führen räumliches Upsampling und Downsampling mittels Faltungsschichten mit Schrittweite und transponierten Faltungen durch. Für tiefere Octrees kann die Struktur vertikal beliebig erweitert werden.

(b) Struktur des OResNet/2 Encoder Blocks

(c) Struktur des OResNet*2 Decoder Blocks. P bezeichnet *Pruning*, das Entfernen von ausgewählten Merkmalsvektoren aus dem Tensor.

Bild 1: Struktur des verwendeten Netzes

dieser Zelle verringern würde. Umgekehrt wird auch bei einer als leer bewerteten Zelle, die in der Ground Truth eine Klasseninformation enthält, ebenso kein geeigneter Gradient bestimmt, da der Berechnungsgraph des Netzes eine Unterbrechung aufweist. In den hier gezeigten Experimenten wird für diese Fälle ein zusätzlicher Kostenterm hinzugefügt, der die Struktur des prädizierten Octrees explizit berücksichtig. Um weiterhin während des Trainings immer ein strukturell mit der Ground Truth übereinstimmendes Ergebnis zu erhalten, wird vom Netz zwar eine Strukturprädiktion vorgenommen, statt dieser jedoch die Struktur der Ground Truth auf den Ausgabe-Octree angewendet. Dies ist auch aus einem weiteren Grund notwendig: Das Netz kann, insbesondere am Anfang des Trainings, die Belegung des Ausgabe-Octrees überbewerten und dadurch einen stark erhöhten Speicherbedarf aufweisen, solange die Strukturprädiktion nicht konvergiert ist. Als Optimierer wird Adam [5] mit entkoppeltem Gewichtsverfall [6] als Regularisierungsmechanismus verwendet. Zusätzlich

findet auch Dropout [10] Anwendung. Aufgrund der spärlichen und unter Umständen stark abweichenden Besetzung der im Netz verwendeten Tensoren wird für die Normalisierung auf eine Instance Normalization [13] zurückgegriffen. So wird der Normalisierungsterm nicht stark von einzelnen Elementen des Minibatches beeinflusst.

3.2 Erzeugung des Eingabe-Octrees

Die Eingabestruktur wird aus einer dreidimensionalen Lidar-Punktwolke erzeugt, indem Zellen, die mehr als einen Punkt enthalten, weiter unterteilt werden, bis nur noch ein Punkt in einer Zelle liegt oder die maximale Tiefe des Octrees $d_{\max}$ erreicht ist. Für die Ground Truth wird die gleiche Methode angewendet, der Octree wird anschließend jedoch gekürzt. Unterteilte Octree-Zellen, die nur Punkte der gleichen Klasse enthalten, werden zusammengeführt und die neu entstandene Blattzelle erhält die gemeinsame Klasse. Dies kann die benötigte Rechenzeit des Netzes reduzieren, wenn große Bereiche der Umgebung zur gleichen Klasse gehören. Die Netzschichten der tieferen Octree-Ebenen können so die Klassifikation von typischerweise kleineren Klassen übernehmen. $d_{\max} = 12$ wird für alle Experimente in diesem Beitrag verwendet, was zu einer maximalen Voxelanzahl von $\left(2^{12}\right)^3 = 68.719.476.736$. Mit einer zusätzlichen Merkmalsdimension im Tensor, die eine Mindestgröße von 10 oder mehr Merkmalen enthält und unter Annahme von 16-Bit-Gleitkommazahlen, die in der Netzarchitektur Verwendung finden, sind dicht besetzte Tensorrepräsentationen auf aktueller Hardware nicht realisierbar. Der Speicherbedarf für die beschriebene Repräsentation läge hier bei über 80 GiB. Daher werden im Eingabe-Octree nur Voxel berücksichtigt, die belegt sind. Die Voxelanzahl auf der tiefsten Ebene des Octrees liegt mit etwa 10^4 in einer auf aktueller Hardware verwendbaren Größenordnung.

3.3 Vorgehen beim Training

Da die prädizierte Struktur des Ausgabetensors sich von der Ground Truth unterscheiden kann, muss dies von der Kostenfunktion berücksichtigt werden.

Tabelle 1: Trainingsparameter

Parameter	Lernrate	Gewichtsverfall	Batchgröße
Wert	0.001	0.0001	12

Zu diesem Zweck werden zwei Kostenterme zum Klassifikationsterm hinzugefügt:

$$\mathscr{L} = \mathscr{L}_{\text{class}} + \mathscr{L}_{\text{active}} + \mathscr{L}_{\text{inactive}} \tag{1}$$

$$\mathscr{L}_{\text{class}} = -\sum_{d=1}^{d_{\max}} \frac{1}{\left|\mathbf{C}_{\text{class},d}\right|} \sum_{\mathbf{c}_{x,y,z} \in \mathbf{C}_{\text{class},d}} \sum_{c_{x,y,z} \in \mathbf{c}_{x,y,z}} \log(c_{x,y,z}) \cdot y^{\star}_{x,y,z} \tag{1a}$$

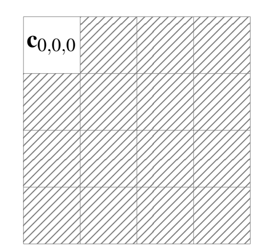
$$\mathscr{L}_{\text{active}} = \sum_{d=1}^{d_{\max}} \sum_{c \in \mathbf{C}_{\text{pred}} \setminus \mathbf{C}_{\text{gt}}} a(\mathbf{c}_d) \tag{1b}$$

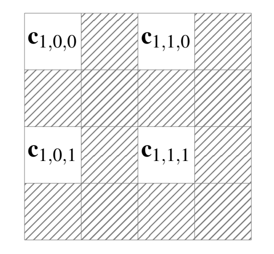
$$\mathscr{L}_{\text{inactive}} = -\sum_{d=1}^{d_{\max}} \sum_{c \in \mathbf{C}_{\text{out}}} a(\mathbf{c}_d) \tag{1c}$$

$\mathscr{L}_{\text{class}}$ ist der übliche Cross-Entropy-Kostenterm, der hier jedoch nur auf Blattknoten des Ground-Truth-Octrees angewendet wird. $\mathscr{L}_{\text{active}}$ bildet die Summe der Aktivitätsfunktion ($a(\mathbf{c}_d)$) von jeder Zelle, die vom Netz als aktiv bewertet wurde, aber nicht zur Ground Truth gehört. Im Gegensatz dazu bestehen die Inaktivitätskosten $\mathscr{L}_{\text{inactive}}$ aus der Summe der Aktivitätsausgabe von jeder Zelle, die in der Ground Truth aktiv ist, unabhängig von ihrem Zustand in der Prädiktion des Netzes. Diese Kostenterme bestrafen somit Strukturunterschiede zur Ground Truth.

Während der Validierung in den ersten Trainingsepochen bleibt die vorgegebene Ground-Truth-Struktur in Verwendung, da vor Konvergenz der Strukturprädiktion die Anzahl an aktiven Zellen zu rapide anwachsendem Speicherbedarf führen kann.

Für die Implementierung des Netzes werden Pytorch [7] und Minkowski Engine [2] verwendet. Tabelle 1 stellt die verwendeten Trainingsparameter dar. Die hierarchische Struktur des Octrees $O = (\mathbf{C}_1, \ldots, \mathbf{C}_{d_{\max}})$ mit der maximalen Tie-





$\mathbf{c}_{2,0,0}$	$\mathbf{c}_{2,1,0}$	$\mathbf{c}_{2,2,0}$	$\mathbf{c}_{2,3,0}$
$\mathbf{c}_{2,0,1}$	$\mathbf{c}_{2,1,1}$	$\mathbf{c}_{2,2,1}$	$\mathbf{c}_{2,3,1}$
$\mathbf{c}_{2,0,2}$	$\mathbf{c}_{2,1,2}$	$\mathbf{c}_{2,2,2}$	$\mathbf{c}_{2,3,2}$
$\mathbf{c}_{2,0,3}$	$\mathbf{c}_{2,1,3}$	$\mathbf{c}_{2,2,3}$	$\mathbf{c}_{2,3,3}$

Bild 2: Struktur des Minkowski-Engine-Tensor-Tupels zur Darstellung eines Quadtrees der Tiefe 3. Schraffierte Zellen werden aufgrund der Schrittweite des betreffenden Tensors niemals belegt.

fe $d_{\max}$ wird als $d_{\max}$-Tupel von spärlich besetzten vierdimensionalen Tensoren $\mathbf{C}_d \in \mathbb{R}^{X \times Y \times Z \times F_d}$ mit $F_d = \min(2^{d_{\max}-d}, 256)$ dargestellt. Der in der Minkowski Engine üblichen Vorgehensweise entsprechend werden die räumlichen Dimensionen als spärlich besetzt behandelt, während der Merkmalsvektor an den besetzten Stellen eine dichte Darstellung besitzt. Die Darstellung des Tensors $\mathbf{C}_d$ im Speicher ist daher ein Tupel der Koordinaten der n aktiven Zellen $\mathbf{K} \in \mathbb{R}^{n \times 4}$ und der zugehörigen Merkmalsvektoren $\mathbf{F}^{n \times F_d}$. Die einzelnen spärlich besetzten Tensoren der Hierarchieebenen teilen sich in der hier verwendeten Implementierung ein gemeinsames diskretisiertes Koordinatensystem, besitzen jedoch abhängig von der Hierarchieebene unterschiedliche Schrittweiten. Für jede höhere Hierarchieebene wird die Schrittweite s des Tensors um den Faktor zwei erhöht. Abbildung 2 zeigt die Struktur der Tensoren am Beispiel eines Quadtrees. Für die dreidimensionalen Faltungsschichten gelten die gleichen Beschränkungen, was effektiv zu einer Dilatation der Kernels führt. Eine Faltungsschicht, die auf einen Tensor mit einer Schrittweite $s_{\mathrm{T,in}} \neq 1$ angewendet wird, muss nicht zwangsweise eine Schrittweite $s_{\mathrm{conv}} \neq 1$ aufweisen, verändert die Schrittweise des Tensors jedoch entsprechend $s_{\mathrm{T,out}} = s_{\mathrm{T,in}} \cdot s_{\mathrm{conv}}$. So entsteht ein hierarchisches Netz, das den in der Bildverarbeitung üblichen U-Nets [9] ähnelt. Von der Architektur in [2] unterscheidet sich das Netz durch die Möglichkeit der Ausgabe von semantischen Informationen auf höheren Hierarchieebenen. Das so entstehende Octree ResNet (OResNet) kann als Kombination von *Shape Reconstruction* [12] und semantischer Segmentierung verstanden werden. Es kann für zusätzliche Aufgabenfelder erweitert werden.

4 Ergebnisse der Lidar-Segmentierung

Um die Güte des beschriebenen Modells zu bewerten, wird eine Auswertung bei einer Punktwolkensegmentierung vorgenommen. Der nuScenes-Lidarseg-Datensatz [1] ermöglicht die Untersuchung des Netzes auf Realdaten. Der Datensatz enthält Punktwolken von Straßenszenen, die mit einem Lidarsensor auf dem Dach eines Fahrzeugs aufgenommen wurden. Die Anwendung eines Netzes, das Octree-Strukturen segmentiert, bildet die Aufgabe der dreidimensionalen Kartierung einer Umgebung ab. Die Güte des Netzes wird, um die für nuScenes-Lidarseg übliche Metrik der *Intersection over Union* (IoU) anwenden zu können, durch anschließende Projektion der Eingabepunktwolke in den Ausgabe-Octree bestimmt. Insgesamt werden die Punktwolken in 32 Klassen eingeteilt, da aber von einigen Klassen nur wenige Beispiele vorhanden sind, werden diese Klassen für die Auswertung der Lidarseg-Challenge entsprechend zu 16 Klassen zusammengeführt. Um die vom Netz prädizierten voxelgenauen Klassen auf die Punktwolke abzubilden, wird diese in den Octree projiziert. Die Klassenprädiktion für einen einzelnen Punkt entspricht dabei der Klasse, die das Netz für den Blattknoten, in dem dieser Punkt liegt, bestimmt hat. Die Klassifikationsergebnisse werden dann auf dem nuScenes LidarSeg-Validierungsdatensatz bewertet, da für das Testset keine Ground Truth vorliegt. Tabelle 2 zeigt die Ergebnisse der punktweisen Segmentierung auf dem nuScenes LidarSeg-Datensatz für aktive und inaktive Strukturinferenz.

Hier zeigt sich ein deutlicher Einfluss der Strukturinferenz durch das Netz auf das Endergebnis der Segmentierung. Die integrierte Strukturschätzung zeigt im Vergleich zur Vorgabe der räumlichen Struktur der Ground Truth verringerte Werte für die IoU. Die räumliche Ausdehnung der zu detektierenden Klasse beeinflusst die Detektionsgüte ebenfalls. Klassen, die üblicherweise ein größeres Volumen einnehmen, werden vom Netz zuverlässiger korrekt klassifiziert. Für einzelne Klassen von kleineren Objekten wie der Klasse *Motorcycle* zeigt sich dagegen eine verbesserte Prädiktionsgüte bei der Strukturprädiktion. Dies deutet auf Vorteile durch die Verwendung der weiteren dem Netz zur Verfügung stehenden Faltungsschichten für die Klassifikation hin.

Tabelle 2: Intersection over Union für die unterschiedlichen Klassen des nuScenes-LidarSeg-Datensatzes

Klasse	Inferenzmethode	
	GT-Struktur	Strukturinferenz
Barrier	0,45	0,31
Bicycle	0	0
Bus	0,24	0,19
Car	0,49	0,45
Constr. Veh.	0,04	0,04
Motorcycle	0,06	0,12
Pedestrian	0,08	0,08
Traffic Cone	0,16	0,12
Trailer	0,23	0,24
Truck	0,36	0,25
Driv. Surf.	0,91	0,75
Flat	0,60	0,43
Sidewalk	0,59	0,47
Terrain	0,62	0,51
Manmade	0,74	0,63
Vegetation	0,72	0,71
mIoU	0,39	0,33

5 Zusammenfassung

Dieser Beitrag stellt eine Architektur für ein neuronales Netz vor, das durch Kombination von semantischer Segmentierung und Shape Reconstruction im Decoder einen Octree mit semantischen Informationen prädizieren kann. Die hierarchische Struktur der Eingabe- und Ausgabedaten enthält zwar nützliche Informationen für nachfolgende Aufgaben, die Erstellung der Eingabestruktur ist jedoch mit einem hohen Berechnungsaufwand verbunden. Dies bedeutet, dass das Verfahren für Online-Aufgaben nur eingeschränkt geeignet ist. Das muss jedoch keine Beschränkung bei Offline-Aufgaben wie der Erstellung dreidimensionaler semantischer Karten von großen Szenen sein. Da das Netz mit dem Ziel trainiert wird, eine möglichst effiziente Repräsentation

der Ausgabe zu generieren, muss hier der erhöhte Rechenaufwand bei der Vorverarbeitung mit den Vorteilen dieser Ausgabeform abgewogen werden.

Darüber hinaus bringt die hierarchische Struktur des Netzes einige Einschränkungen mit sich. Das Netz führt zwangsläufig ein Downsampling bis zur Wurzel des Octrees durch, was für die semantische Segmentierung nicht benötigt wird. Diese Berechnungen müssen jedoch durchgeführt werden, um eine Einschränkung des effektiven Receptive Field durch die üblichen 3×3-Kernelgröße der Residual Layer zu vermeiden.

Da das Netz häufige Klassen zuverlässiger bestimmen kann als seltene, könnten Trainingsmethoden wie CutMix [16] auf diesen Ansatz angewendet werden. Um CutMix sinnvoll auf Octrees anwenden zu können, müsste die Methode auf die Octree-Struktur angepasst werden, da ein wiederholtes Berechnen des Octrees während des Trainings zu einer deutlichen Verlängerung der Trainingsdauer führen würde.

Literatur

[1] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan und O. Beijbom. „nuscenes: A multimodal dataset for autonomous driving“.

[2] C. Choy, J. Gwak und S. Savarese. „ 4d spatio-temporal convnets: Minkowski convolutional neural networks“. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3075–3084, 2019.

[3] K. He, X. Zhang, S. Ren und J. Sun. „Deep residual learning for image recognition“. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778. 2016.

[4] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss und W. Burgard. „OctoMap: an efficient probabilistic 3d mapping framework based on octrees“. 34(3):189–206.

[5] D. P. Kingma and J. Ba. „Adam: A method for stochastic optimization“.

[6] I. Loshchilov and F. Hutter. „ Decoupled weight decay regularization“. In: *International Conference on Learning Representations*. 2019.

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai und S. Chintala. „Pytorch: An imperative style, high-performance deep learning library“. In: H. Wallach, H. Larochelle, A. Beygelzimer, F. Alché-Buc, E. Fox und R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc. 2019.

[8] C. R. Qi, H. Su, K. Mo und L. J. Guibas. „Pointnet: Deep learning on point sets for 3d classification and segmentation“.

[9] O. Ronneberger, P. Fischer und T. Brox. „U-net: Convolutional networks for biomedical image segmentation“. In: *Lecture Notes in Computer Science*, pages 234–241. Springer International Publishing. 2015

[10] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever und R. Salakhutdinov. „Dropout: a simple way to prevent neural networks from overfitting“. *The journal of machine learning research*, 15(1):1929–1958. 2014.

[11] H. Tang, Z. Liu, S. Zhao, Y. Lin, J. Lin, H. Wang und S. Han. „Searching efficient 3d architectures with sparse point-voxel convolution“. In: *European Conference on Computer Vision*. 2020.

[12] M. Tatarchenko, A. Dosovitskiy und T. Brox. „Octree generating networks: Efficient convolutional architectures for high-resolution 3d outputs“. In: *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2107–2115. 2017.

[13] D. Ulyanov, A. Vedaldi und V. Lempitsky. „Instance normalization: The missing ingredient for fast stylization“.

[14] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun und X. Tong. „O-cnn: Octree-based convolutional neural networks for 3d shape analysis“.

[15] Y. Xie, J. Tian und X. X. Zhu. „Linking points with labels in 3d: A review of point cloud semantic segmentation“. *IEEE Geoscience and Remote Sensing Magazine*, 8(4):38–59. 2020.

[16] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe und Y. Yoo. „Cutmix: Regularization strategy to train strong classifiers with localizable features“. In: *Proceedings of the IEEE/CVF international conference on computer vision*, pages 6023–6032. 2019.

Constrained Design of Experiments for Data-Driven Models

Fabian Schneider[1,3], Max Schüssler[1], Ralph Hellmig[2,3], Oliver Nelles[1]

[1]Universität Siegen, Department Maschinenbau,
Institut für Mechanik und Regelungstechnik – Mechatronik
Paul-Bonatz-Str. 9-11, 57068, Siegen, Germany
E-Mail: {fabian2.schneider,max.schuessler,oliver.nelles}@uni-siegen.de

[2]Universität Siegen, Department Maschinenbau,
Lehrstuhl für Materialkunde und Werkstoffprüfung
Paul-Bonatz-Str. 9-11, 57068, Siegen, Germany
E-Mail: ralph.hellmig@uni-siegen.de

[3]EJOT SE & Co. KG
Im Herrengarten 1, 57319, Bad Berleburg, Germany
E-Mail: {fschneider,rhellmig}@ejot.com

Abstract

The quality of data-driven models depends significantly on the data distribution in the input space. In this work, design of experiments (DoE) methods for constrained input spaces are discussed. An approach based on an Latin hypercube (LH) design is introduced to deal with strongly constrained input spaces. For the unconstrained case, where the input space is a hypercube, different design of experiments methods have been developed. The dominating state-of-the-art methods are Sobol sequences and Latin hypercubes. Instead of optimizing complete LH designs, the proposed strategy is to incrementally construct an LH design. Every new sample is selected by a distance-based metric. The presented method is then applied in two test cases and compared to a method based on a Sobol sequence. Here, an initial design is created by a

Sobol sequence and every sample is removed that violates the constraints. The design qualities are measured by the resulting model accuracies of data-driven models. A function generator is applied to create synthetic data sets to train and evaluate local linear model networks.

1 Introduction

Models are a basic component of several advanced techniques in many disciplines. They are utilized in applications like simulation, optimization, control, and fault detection tasks. The requirements for model accuracy and complexity increase in all these applications. As a result, deriving a model by first order principles is often not applicable, because the derivation of such models is often to complicated or the model evaluation is too slow. Data-driven models which rely only on measurements are more and more frequently the better choice. If a model relies solely on measurement data, the importance of good and efficient measurement plans is obvious. In the scope of this work, a measurement can be a real physical measurement or created by computer experiments. For example by the evaluation of first principle models like the finite element analysis. A measurement plan is also called experimental design. To create such a design, different design of experiment strategies have been developed [9].

A standard design space which is also the input space of the model is a d-dimensional hypercube. If no prior knowledge for a process exists, uniformly distributed experimental designs should be used [9]. Sobol sequences are a quite popular choice and an example for a low-discrepancy series. These designs are uniformly distributed over the input space, even for high-dimensional problems. Optimized Latin hypercube (LH) designs achieve also good results in numerous applications. The maximin-criterion is commonly used for optimization [12]. It maximizes the minimal distance between two samples of the design. One drawback of those maximin-LHs is the need for solving an optimization problem. It is hard to solve and also computational expensive. The difficulties increase with the dimensionality and design size. The perfect one-dimensional projection property of LHs is one advantage for modeling tasks. Those two mentioned methods are suited for unconstrained input spaces.

Constraints leads to an unregular d-dimensional input space. As a result new or adapted methods must be applied for such tasks.
The constraints lead also to a conflict of objectives. It is not possible to create an uniformly distributed design over the feasible space with uniformly distributed one-dimensional projection properties. Two different methods are compared in this work to analyze this conflict. The first method is based on a Sobol sequence. After an initial design is created, every sample violating the constraint is removed from the design. This is an easy method. And this should lead to an approximately uniformly distributed design over the feasible regions of the input space. However, depending on the constraints, the projections are not uniformly distributed. The second method is a version of an LH design. In an incremental procedure, new samples are added to the design. Out of a set of feasible new samples, in every iteration the sample is selected by the ϕ_p-criterion [15]. Therefore, the sample is selected that has the largest distance to its nearest neighbors.
A function generator is applied, to create test functions. These are used to build synthetic data sets. The created designs are evaluated by a comparision of the test errors of local linear models for those generated functions.

2 Design of Experiments

Design of experiments covers all methods and strategies to create an experimental plan. In the early 20th century, Fisher [4] introduced a method to investigate statistical properties of crop systems. In recent time, DoE strategies are also applied to plan computer experiments with the goal building a meta or surrogate model. Computer experiments are in most cases deterministic. Computer experiments, like Finite Element Analysis, are often computationally very expensive. So, one target of DoE for computer experiments is to create a design that extracts the necessary information to model the process with as few samples as possible. Passiv learning strategies are the standard case. The design is defined before the training of the model has started or the data have been measured or computed. All regions of the input space are weighted equally. Thus, an uniformly distributed design is desired [9]. Besides passiv learning strategies, active learning strategies have been developed [1]. In

contrast to passiv learning strategies, active learning strategies are incremental. The selection of every new sample or a new batch of samples is for example based on the model quality. Utilizing the model error as criterion increases the sample density in regions of high model errors. The process output could also serve as a criterion. In model based optimization tasks, regions of the input space with undesired process outputs do not need to be modeled with the same accuracy as regions of high interest. Designs created by passive methods typically serve as initialization of active strategies.

Different evaluation metrics of the design distribution have been developed for unconstrained design spaces. Common are discrepancy based methods or metrics, like the centered L2 discrepancy [15] which describes the uniformity of the design over the input space and of all projections over the corresponding subspaces. The Kullback-Leibler-Divergence is also applied frequently. It is a distribution based method. The design is asset by comparing the actual sample distribution with the desired one. As a third group, distance-based metrics exists. Here, the distances between the samples of the design serves as basis for evaluation. The ϕ_p-criterion belongs to this group [15]. Constraints often prevents the application of those standard methods. The estimation of sample distributions over a constrained input space is in complex cases not feasible. Kernel density estimation methods are not suited for unregular input spaces. Boundary effects impair an accurate computation [10].

ϕ_p-criterion

It is not desirable that samples of a design are close to each other, because then, it is expectable that only few new information is generated by each sample. Thus, generally a design with large nearest neighbor distances (NN) for all samples is preferred. A common strategy to achieve large NN distances is to maximize the minimal NN distance. Unfortunately, if a sample is modified to improve the distance to its actual nearest neighbor, it is possible that another sample becomes the new NN. In this case, the NN distance criterion is non-smooth. Therefore, all gradient-based solution strategies are inapplicable. Additionally, only the minimal NN distance is considered by this procedure. All remaining samples are irrelevant.

Therefore, the ϕ_p-criterion has been developed [6] to obtain a smooth approximation of the minimal NN distance criterion:

$$\phi_p(\mathbb{U}) = \left\{ \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} \|\underline{u}_j - \underline{u}_i\|^{-p} \right\}^{1/p}, \tag{1}$$

where $\mathbb{U}$ is the set of samples of the design and each sample is defined as $\underline{u}_i = \left[u_{i,1}\ u_{i,2}\ \cdots\ u_{i,d}\right]^{\mathrm{T}}$. While the minimal NN distance has to be maximized, the ϕ_p criterion has to be minimized. The price to be paid is a higher computational load because all sample pairs have to be evaluated in (1) and no efficient K-D Tree search algorithms can be utilized. This formulation is better suited for optimization tasks. The weighting of the nearby samples increases with growing values for p. Commonly used values are $p = 15 \ldots 50$ [6]. In this work, all studies have been carried out with $p = 15$.

2.1 Sobol Sequence

Sobol proposed this sequence [11] in 1967. It is a low-discrepancy, quasi-random design and is commonly applied for modeling tasks. Numerical integration tasks are a further common application. Compared to modeling tasks, the dimensionality of integration problems is often larger. As a result, Sobol provided sequences for up to 51 dimensions. This sequence is designed to create samples as uniformly distributed as possible over the input space [11].

2.2 Latin Hypercube Sampling

Latin hypercube designs are constructed to ensure perfect one-dimensional projection properties. LHs are per definition non-collapsing designs. To create an d-dimensional Latin hypercube with N samples, each dimension is divided into N levels. Each sample of the design is, as originally proposed, created by a random combination of the available levels in each dimension.
By this procedure, designs may occur that have poor space filling qualities [12]. To avoid such designs, optimization strategies have been developed. The resulting optimization problems are hard to solve. An overview over

different methods is given in [12]. In [3], an LH design is optimized by coordinate swapping. The method is called Extended Deterministic Local Search algorithm (EDLS). The optimization target is to create a maximin design by optimizing the ϕ_p-criterion, see (1), of the design $\mathbb{U}$. The EDLS method outperforms many other LH designs in the unconstrained case. Especially, the translational propagation (TPLHD) algorithm [13] and the iterated local search (ILS) algorithm [5].
In the unconstrained case a coordinate swapping is always possible. In the constrained case, both modified samples have to satisfy all constraints after the swapping. If the input space is highly restricted, most swapping operations are infeasible which leads to significantly poorer design qualities.

2.3 Incremental Latin Hypercube

The incremental Latin hypercube (ILH) algorithm is proposed in the following. In [14], an incremental methodology is presented to build LHs. Here, the dimensionality of the design and the design size grow in each increment.
The in this work presented method constructs a design in an incremental way, instead of modifying a complete LH design. In every iteration, a new sample is added to the design. As a result, no coordinate swapping is necessary. No global optimization is performed.
Similar to the standard LH designs, the interval of each dimension is divided into N levels. In every iteration, one sample is added to the design. The levels corresponding to this sample are afterwards removed from the set of available levels. As described in algorithm 1, in each iteration a set of possible new samples based on the remaining grid levels is randomly created. The sample of that set is selected that achieves the best ϕ_p value.
For an example of the algorithm see Fig. 1. The target design size is $N = 5$. In every iteration $m = 3$ candidate samples are tested. The constraint, the unfeasible input space, is marked by gray color. The samples of the design at each increment are represented by the crosses. The free levels are marked by dashed gray lines, the already used levels by gray lines, and the levels of the selected point by dotted black lines. The five increments are:

a) Randomly chosen initial point.

Algorithm 1.: Incremental Latin Hypercube (ILH)

Input: N: design size, d: dimensionality, $\mathbb{C}$: set of constraints,
m: number of candidate samples in each iteration

Output: $\mathbb{U}$: generated design

1: Create for every dimension k a set $\mathbb{G}_k$ with N levels (default: equally spaced) in $[0, 1]$
2: Initialize the design $\mathbb{U}$ with a random sample $\underline{u}_1$ by selecting one level of each set $\mathbb{G}_k$. This sample has to fulfill the constraints in $\mathbb{C}$. Until this sample fulfills the constraint, step 2 is repeated.
3: Remove the chosen levels from each set $\mathbb{G}_k$.
4: **for** $i = 2$ to N **do**
5: Build set $\mathbb{S}$ containing m candidate samples by choosing random levels in $\mathbb{G}_k$.
6: Remove all samples of $\mathbb{S}$ that violates any constraint in $\mathbb{C}$.
7: **if** $\mathbb{S} \neq \emptyset$ **then**
8: For every element $\underline{s}_j$ of $\mathbb{S}$, calculate $\phi_{p,j}$ of the design $\mathbb{U} \cup \underline{s}_j$, for $j = 1, \ldots, \#\mathbb{S}$.
9: $\tilde{\underline{s}}$: Sample of $\mathbb{S}$ which achieves best ϕ_p value, $\min\limits_j \phi_{p,j}$
10: $\mathbb{U} := \mathbb{U} \cup \tilde{\underline{s}}$.
11: Remove each level of $\tilde{\underline{s}}$ from the corresponding set $\mathbb{G}_k$, for $k = 1, \ldots, d$.

b) First iteration, triangle, circle, and square are candidate samples. Square lies in the unfeasible region. Triangle achieves lowest ϕ_p value, see Tab. 1. It is added to $\mathbb{U}$.

c) Second iteration, triangle is selected by ϕ_p value and added to $\mathbb{U}$.

d) Only 2 levels are left in each set of levels $\mathbb{G}_k$. Therefore, one level is selected 2 times ($m = 3$). Only the triangle fulfill the constraint. It is selected.

e) For each dimension, only one free level is remaining. Thus, only one candidate sample is created. It violates the constraint. Thus, no sample is added to the design $\mathbb{U}$.

Finally, instead of the desired $N = 5$ samples, the design contains only 4 samples. In this case, it is possible to place all desired 5 samples in the feasible input space. For example, it is possible to place all points on the diagonal line. Please note, that this isn't the normal case. But as illustrated by this example, it is obvious that depending on the selections at earlier increments

Table 1: ϕ_p-values of the candidate samples

Iteration	Triangle	Circle	Square
a)	–	–	–
b)	2	4	unfeasible
c)	4	4	unfeasible
d)	4	unfeasible	unfeasible
e)	unfeasible	–	–

the resulting design size may vary. A determination of the exact design size priorily isn't possible. This algorithm requires only one hyper parameter. It is the parameter m. It have to be defined depending on the dimensionality and the desired design size. While larger values improve the design quality, they also lead to a rise of the computational costs, because more constraint and ϕ_p evaluations are necessary.

3 Function Generator

The evaluation and comparison of different methods in the constrained case is challenging. The comparison in this work is based on test functions. In [2] a function generator is proposed. It is based on polynomial functions and applicable for different dimensionalities. The complexity of the functions can be controlled by the user. The input space of the function is the d-dimensional hypercube. The functions are generated according to

$$g(u_1, u_2, \ldots, u_d) = \sum_{i=1}^{M} c_i \cdot (u_1 - s_{i,1})^{p_{i,1}} \cdot (u_2 - s_{i,2})^{p_{i,2}} \cdot \ldots \cdot (u_d - s_{i,d})^{p_{i,d}}. \quad (2)$$

All parameters are drawn randomly from probability distributions. The coefficients c_i are drawn from a normal distribution $N(0,1)$, the shifts $s_{i,j}$ from the uniform distribution $U(0,1)$ and the powers $p_{i,j}$ from an exponential distribution with expected value μ and are rounded down [2]. M defines the number of monomials or terms. This and μ controls the complexity of the generated

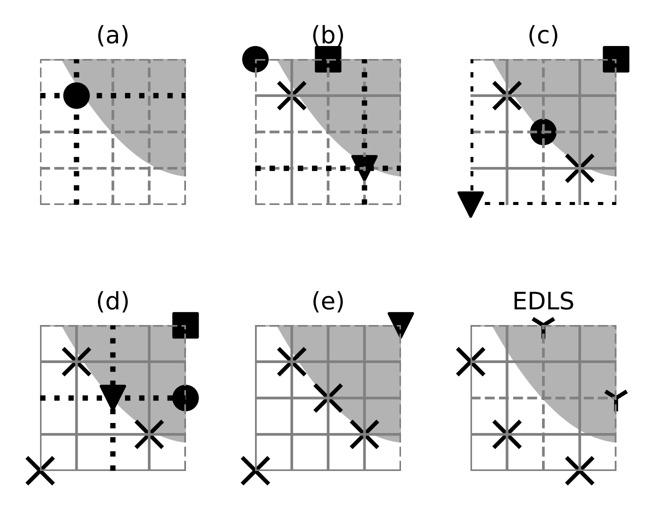


Figure 1: A 5 × 5 example of the ILH algorithm. Plots (a) to (e) describe the five increments. The last plot shows an via EDLS optimized LH. Here, tow samples are unfeasible.

functions. In the scope of this work, all functions are created with $M = 10$ and $\mu = 2$. This parameter combination results in complex functions. Some examples are shown in Fig. 2. The function outputs are normalized to the interval $[0, 1]$.

4 Results

First, the unconstrained case is used to study some properties of the different methods. With these insights the ILH is applied to 2 different test constraints. The first one is a two-dimensional problem. The input space is constrained by a polynomial function. The resulting shape is comparable to an efficiency map of a combustion engine with the two inputs rotation speed and torque. The second application is a parametrization of a geometrical body. The body is parameterized by five parameters. Thus the input space is 5-dimensional. The volume of the generated solids has to be constant. Only small tolerances are allowed. So, not all parameter combinations are feasible. Such a constraint or

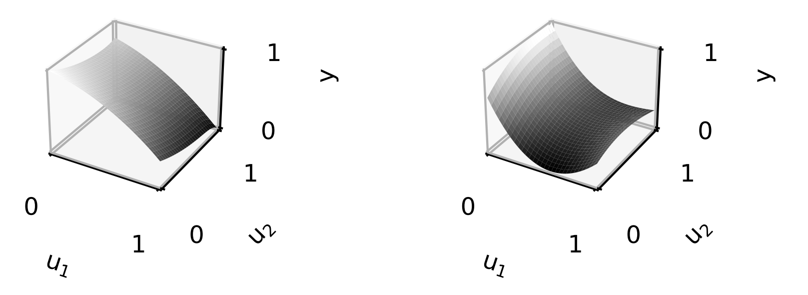


Figure 2: Two two-dimensional normalized functions generated with $M = 10$ and $\mu = 2$.

similar ones could occur for example in forming processes. Here, the volume constancy of the processes restricts the valid shapes of the solids.

4.1 Studies of the Unconstrained Case

To understand the constrained case easier, some properties of the different design methods and the ϕ_p-criterion, see (1), are analyzed.
ϕ_p is often selected as a criterion in LH optimizations. It drives the samples as far away from each other as possible. This behavior is shown by the following example. Based on a Sobol sequence $\mathbb{U}_{sob}$, a test design $\mathbb{U}_\phi$ is created. The first data sample of $\mathbb{U}_{sob}$ initializes $\mathbb{U}_\phi$. In an incremental procedure, beginning from the second sample of $\mathbb{U}_{sob}$, every sample $\underline{u}_i$ is optimized. The objective is to minimize ϕ_p of the merged design $\mathbb{U}_\phi \cup \underline{u}_i$ according to

$$\begin{aligned} &\min_{\underline{u}_i} \phi_p(\mathbb{U}_\phi \cup \underline{u}_i) \\ &\text{subject to} \\ &0 \leq u_{i,k} \leq 1,\ k = 1 \ldots d. \end{aligned}$$

The solution of the optimization problem, the resulting sample, is added to $\mathbb{U}_\phi$. Afterwards these steps are repeated with all remaining samples. Each sample of $\mathbb{U}_\phi$ is placed, such that it minimizes ϕ_p at the current iteration. This is similar to a maximization of the distance to the nearest neighbor of $\underline{u}_i$. The one- and

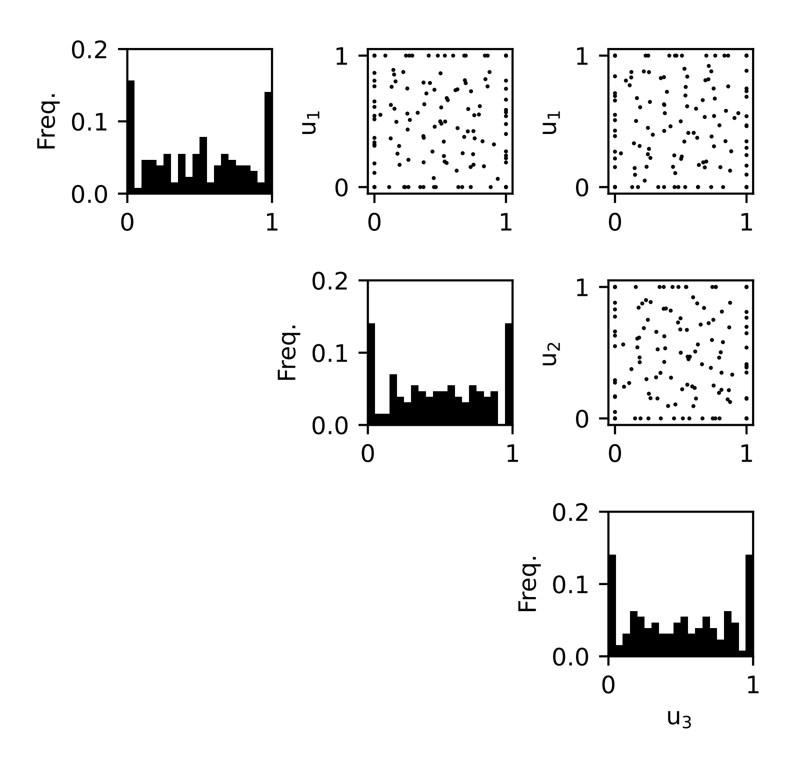


Figure 3: Projections of a three-dimensional design with 128 samples incrementally optimized with regard to the ϕ_p-criterion.

two-dimensional projections of such a three-dimensional design are shown in Fig. 3. Obviously, the optimization drives the samples near to the boundaries of the hypercube. The one-dimensional projections are not uniformly distributed at all. According to [2] a design with samples in the edges and, corners of the input space is only preferable if large extrapolation is demanded. The curse of dimensionality implies that with increasing dimensionality a larger volume of the input space is near to the boundaries of the hypercube. Thus, the behavior already observed in the three-dimensional case for the one-dimensional projections would further increase with the dimension. The LH design enforces per definition a uniform distribution of the one-dimensional projections. This

could be interpreted as a regularization of the ϕ_p-criterion. Accumulations of samples close to the boundaries are prevented.
Based on this observations, it is expectable that the nearest neighbor (NN) distance of each sample grows with the distance to the center of the hypercube. In Fig. 4 the nearest neighbor distance of each sample is shown for different distances to the center of the input space. The distances are normalized by the maximal possible distance, the distance from the center of the hypercube to a corner. A Sobol sequence, a random LH design, the presented ILH design and a via the EDLS algorithm [3] optimized LH are compared. The results of the Sobol sequence and the random LH are as expected. The nearest neighbor distance grows with increasing distance to the center. Also the variance in each group is comparably large. The EDLS design instead surprises. The variance of the nearest neighbor distance is smaller. This is a result of the optimization. But, not expected, the nearest neighbor distances over all intervals are nearly constant and overall larger compared to the other methods. This might be another reason besides slightly better one-dimensional projection properties why EDLS optimized LHs are preferable for modeling tasks compared to the Sobol sequences [3]. This property might be even more important for kernel-based models like Gaussian process regression models. The ILH design achieves also better results as the Sobol sequence overall. Compared to the EDLS design, the NN distances are slightly smaller and the variance is larger.

4.2 Parabola Constraint

In this section, the ILH design is applied to a single constraint in 2D. It is motivated from typical maps of combustion engines with maximum power constraints. All samples i in the unit hypercube that also fulfill

$$-(1.3 \cdot (u_{i,1} - 1)^2 + 0.2 - u_{i,2}) \leq 0 \tag{3}$$

are feasible design points.
In Fig. 5 a Sobol sequence (all unfeasible samples are removed), and two ILH designs are presented. An EDLS design is not applicable because the constraint prohibits most of the swapping options. This makes an optimization unfeasible. For both strategies, the resulting number of samples is not known

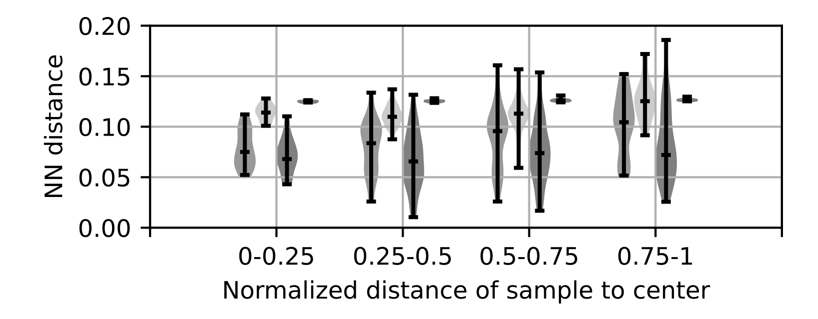


Figure 4: Nearest neighbor distances of every sample over the distance of the sample to the center. Each design is three-dimensional and consists of 512 samples. From left to right: Sobol sequence, ILH, random LH, EDLS optimized LH

exactly before the construction of the designs. The sizes of the initial designs are chosen such, that the resulting design has a size of 95 % to 100 % of the target size.

To compare the quality of the created designs, data-driven models are trained. The function generator is applied to create 20 different artificial processes. A local linear model network with an incremental tree construction algorithm (HILOMOT = Hierarchical Local Model Tree) [7] is trained for every process. In this study, 10 different ILH designs are evaluated. In Fig. 5 two out of those 10 ILH designs and the design based on the Sobol sequence are shown. Designs with sizes from 64 to 2 048 are created. The test errors of the trained models are presented in Fig. 6. In total, the ILH design achieved better results than the Sobol sequence in 58% of the 1 200 different ILH designs. For two exemplary models, the absolute values of model errors are shown in Fig. 7. The largest error occurs in the lower left corner of the model trained with the Sobol sequence. In accordance with the results of the comparison of the nearest neighbor distances it is expected that more samples of the ILH designs are close to the boundaries of the input space. In these regions, the model quality with the ILH is generally higher.

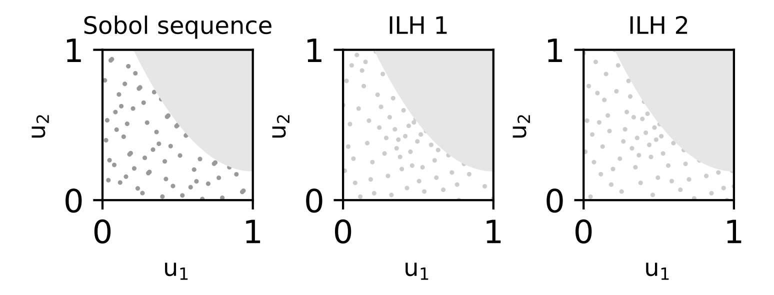


Figure 5: Different designs created, each contains 128 samples. Sobol sequence (left one) and two different ILH designs.

4.3 Volume Constancy

In the second test case, a 5-dimensional and constrained input space is analyzed. The five inputs describe the shape of a rotationally symmetric solid. Basis splines (B-splines) are applied to describe the contour of the part, see Fig. 8. For more information on B-splines see for example [8]. A parametrization is necessary to perform a geometry optimization. In the field of forming or forging processes the volume of the created parts is often restricted. It has to be constant over the forming or forging process. This requirement restricts the allowed parameter combinations significantly. In the test case, the volume tolerance is $\pm 2\,\%$. In Fig. 9 two-dimensional projections of the constrained design space are illustrated. The solid is manipulated by the position of the marked control points (CP) of the B-splines. The assignment with the allowed value ranges of the CPs to the design is given in Tab. 2. The designs for this application are evaluated similarly to the previous. 10 different ILH designs and one Sobol sequence are constructed. 20 different test functions with $M = 10$ and $\mu = 2$ are generated. An overview of the results is given in Fig. 10. In this application, the ILH designs outperform the Sobol sequence in 78% of the cases. The one-dimensional projections of both methods differ significantly, see Fig. 11. The projections of the ILH design are more uniform distributed than the projections of the Sobol sequence. The nearest neighbor distances

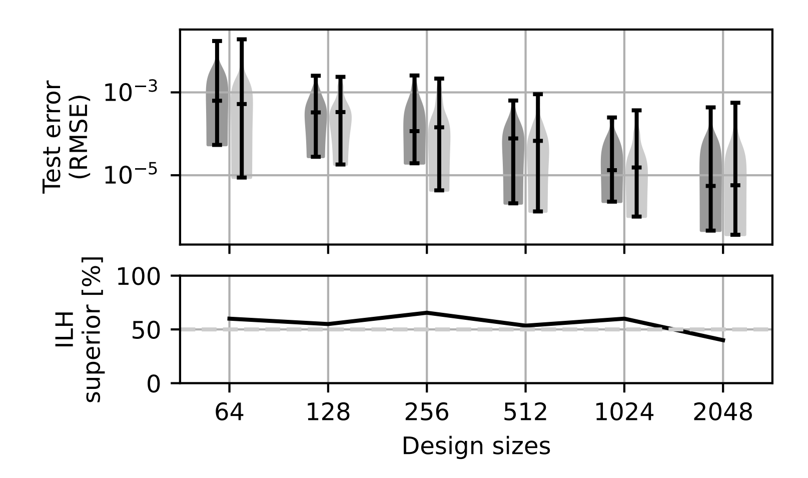


Figure 6: Model errors of all trained models. Dark gray and left: test errors of models trained with Sobol sequence. Light gray and right: test errors of models trained with ILH designs. In the lower plot, the probability is given that a model trained with an ILH design is superior to the model trained with the Sobol sequence.

are also larger for the ILH designs. These results match the observations of the unconstrained case. They suggest that the one-dimensional projection properties has a significant influence on the model quality.

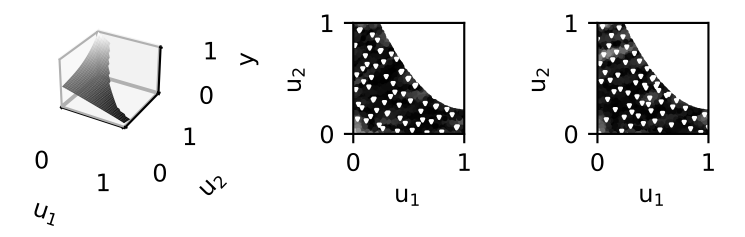


Figure 7: Left plot: test process generated by function generator. Centered plot: Head map of absolute model errors of a model trained with a design based on a Sobol sequence. Right plot: Head map of absolute model errors of a model trained with an ILH design. The samples of each design are visualized by the white triangles.

Table 2: Parametrization of the geometric solid

	Control point	Coordinate	Min	Max
u_1	0	y	3.7	4.7
u_2	1	x	3.9	5
u_3	2	x	4	6
u_4	1	y	1.5	4
u_5	2	y	0.5	2.4

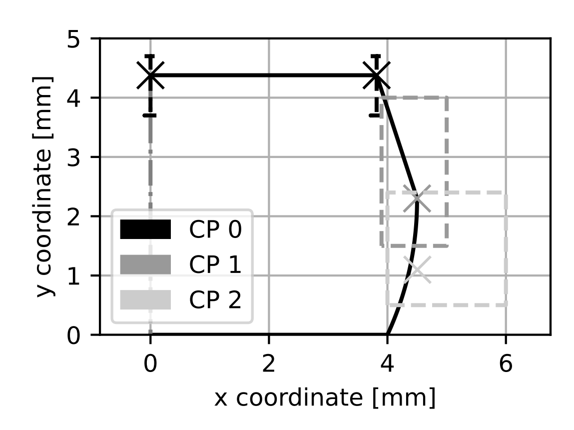


Figure 8: Parametrization of the test case. Parameterized control points with allowed coordinate ranges are shown.

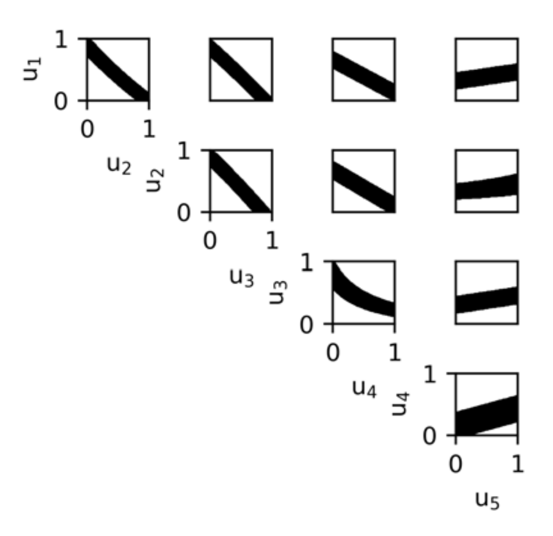


Figure 9: Two-dimensional projections of the input space based on the center point of the input space. Black areas represents the feasible region.

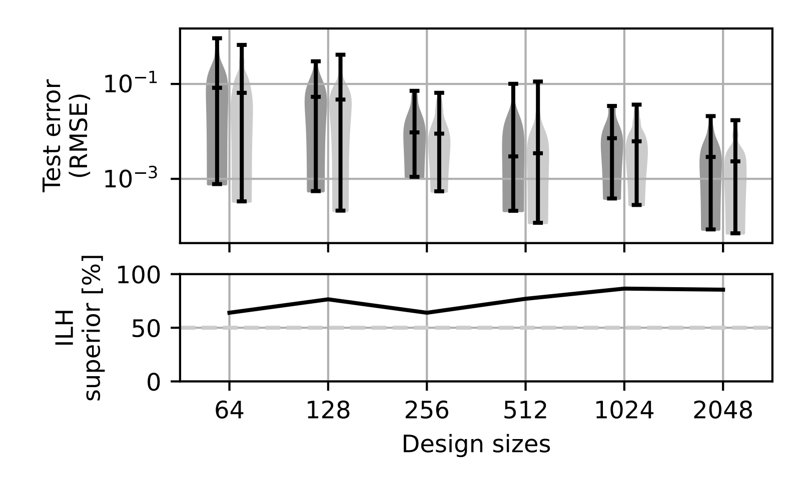


Figure 10: Model errors of all trained models. Dark gray and left: test errors of models trained with Sobol sequence. Light gray and right: test errors of model trained with ILH designs. In the lower plot, the probability is given that a model trained with an ILH design is superior to the model trained with the Sobol sequence.

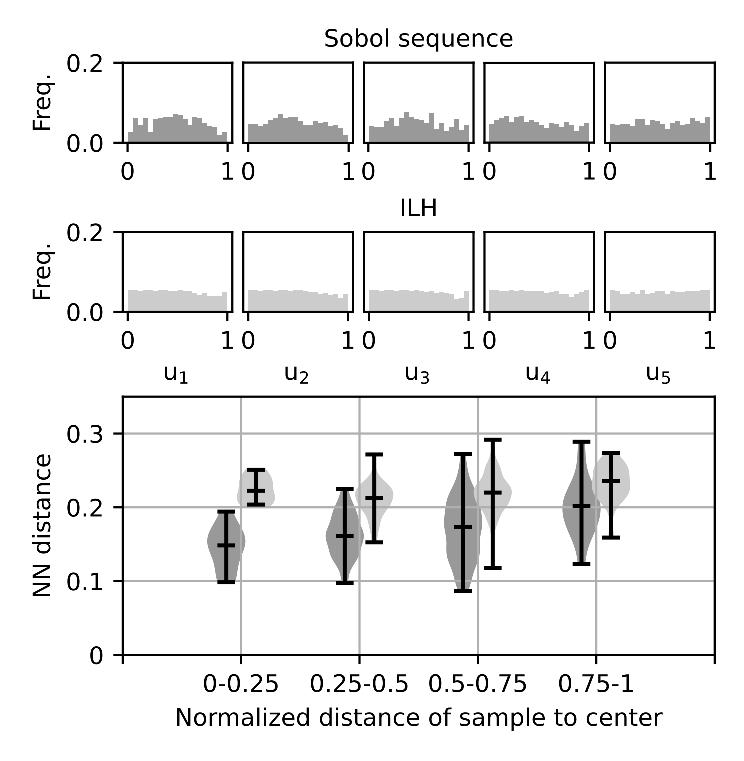


Figure 11: One-dimensional projections and nearest neighbor distances (Sobol sequence left, ILH right) of the Sobol sequence and a randomly selected ILH design with 512 samples.

5 Conclusion

Two algorithms for design of experiments for constrained input spaces are compared in this work. They have been applied to two test processes, a two-dimensional and a 5-dimensional. The quality of the created designs has been evaluated on the basis of the test error of one data-driven models. Due to the limitations of different evaluation criteria for constrained design of experiments, a function generator is applied to create test processes for the evaluation of different designs. A novel strategy to create Latin hypercubes for constrained design spaces has been proposed. The algorithm is based on an incremental construction strategy. In each increment, the best sample is added to the design. This is selected by the ϕ_p-criterion from a randomly generated set of candidate samples. These proposed incremental Latin hypercube (ILH) designs achieve lower model errors for both applications. The one-dimensional projections of the ILH designs are more uniformly distributed than the projections of the design based on Sobol sequences. The nearest neighbor distances of the ILH designs are also higher compared to Sobol sequences.

References

[1] J. Belz. *Fighting the curse of dimensionality with local model networks*. PhD thesis, Universität Siegen, 2018.

[2] J. Belz and O. Nelles. Proposal for a function generator and extrapolation analysis. In *2015 International Symposium on Innovations in Intelligent SysTems and Applications (INISTA)*, pages 1–6, 2015.

[3] T. Ebert, T. Fischer, J. Belz, T. O. Heinz, G. Kampmann, and O. Nelles. Extended deterministic local search algorithm for maximin latin hypercube designs. In *2015 IEEE Symposium Series on Computational Intelligence*, pages 375–382, 2015.

[4] R. A. Fisher. The arrangement of field experiments. 1992.

[5] A. Grosso, A. Jamali, and M. Locatelli. Finding maximin latin hypercube designs by iterated local search heuristics. *European Journal of Operational Research*, 197(2):541–547, 2009.

[6] M. D. Morris and T. J. Mitchell. Exploratory designs for computational experiments. *Journal of Statistical Planning and Inference*, 43(3):381–402, 1995.

[7] O. Nelles. Axes-oblique partitioning strategies for local model networks. In *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, pages 2378–2383, 2006.

[8] L. Piegl and W. Tiller. *The NURBS Book*. Springer Berlin Heidelberg, 1995.

[9] L. Pronzato and W. G. Müller. Design of computer experiments: space filling and beyond. *Statistics and Computing*, 22(3):681–701, apr 2011.

[10] D. W. Scott. *Multivariate Density Estimation*. Wiley, aug 1992.

[11] I. Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 7(4):86–112, 1967.

[12] F. Viana. Things you wanted to know about the latin hypercube design and were afraid to ask. 05 2013.

[13] F. A. C. Viana, G. Venter, and V. Balabanov. An algorithm for fast optimal latin hypercube design of experiments. *International Journal for Numerical Methods in Engineering*, 82(2):135–156, 2010.

[14] T. Voigt, M. Kohlhase, and O. Nelles. Incremental latin hypercube additive design for lolimot. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1, pages 1602–1609. IEEE, 2020.

[15] M. Vořechovský and J. Eliáš. Modification of the maximin and ϕ_p (phi) criteria to achieve statistically uniform distribution of sampling points. *Technometrics*, 62(3):371–386, 2020.

Spatial Control in Model-Based Neural Style Transfer

Julian Bültemeier, Christoph-Alexander Holst, Volker Lohweg

inIT – Institute Industrial IT
Technische Hochschule Ostwestfalen-Lippe,
Campusallee 6, D-32657 Lemgo, Germany
E-Mail: {julian.bueltemeier, christoph-alexander.holst,
volker.lohweg}@th-owl.de

1 Introduction

Neural Style Transfer (NST) is an optimisation technique that combines two images – a content image and a style image. The result of the NST is an output image that looks like the content image but is rendered in the style of the style reference image [1]. An example of such a stylisation is shown in Figure 1. The application areas for such a method are manifold. Examples include image and video editing software [2, 3] and virtual reality applications [4].

The style transfer with the original formulation is a gradient-based optimisation procedure [1]. This procedure can take two to three minutes for the stylisation of a single image. For this reason, methods have been developed to speed up this process [5, 6, 7]. The idea of these model-based NST approaches is to train a feed-forward neural network to learn a direct conversion from a content image to an NST result image. After successful training, it is possible to perform the stylisation in a feed-forward pass without optimisation procedures. There are model-based NST methods that are limited to learn a single style [5, 6, 7], and there are Multi Style Transfer (MST) methods that can learn multiple styles [8, 9, 10]. However, although MST techniques can learn multiple styles simultaneously, they do not integrate spatial control or integrate it only after the style has been trained on complete images. The idea



Figure 1: Example of a Neural Style Transfer with the content image (left), the style image used "*Wheat Field with Cypresses*" by Vincent van Gogh (centre) and the result of the NST (right).

of spatial control in NST is that images often consist of smaller segmentations that correspond to individual objects or further subdivisions. These regions have their own sub-styles [11]. This means that the style within an image differs in local regions. For this reason, NST procedures exist in which spatial control is integrated [11, 12]. Such a semantic style transfer makes it possible to explicitly assign the sub-style of a region in the style image to a region in the content image [11]. This control helps to improve the overall result of the style transfer. However, semantic style transfer has so far been studied mainly for optimisation techniques on a single image [11, 12].

The main contribution of this paper is therefore to integrate a spatial control into model-based NST. First, it is shown how existing MST procedures need to be adapted to be able to learn the sub-styles of a style image. In the second part, a concept is proposed to integrate the semantic transfer into the training in order to learn local style representations. In the final part, the proposed concept is evaluated using the intaglio style as an example. The intaglio style is created during intaglio printing and is an essential component on banknotes [13]. In a preliminary work in MEIER et al. it was shown [14] that the style in individual regions such as the *eye*, differs strongly from other regions. Therefore Intaglio Style Transfer (IST) has the potential to profit from a semantic style transfer.

2 Preliminaries

Humans have always been attracted and inspired by the art of painting. The imitation of certain styles require special skills of well-trained artists. Stylisation is a complex image processing task. A machine approach to this process

is described by the seminal work by GATYS et al. [1]. They used an encoder V created by the Visual Geometry Group (VGG) at Oxford University [15] to extract content and style representations from images. The result was a significant increase in performance in automatically creating new images with a given style, compared to traditional synthesis in pixel space [16] or in a hand-crafted feature space [17].

There are mainly two loss functions defined in the NST technique – the *content loss function* $\mathrm{L_C}$ and the *style loss function* $\mathrm{L_S}$ [1]. If the synthesised image $\mathbf{I}$ is to have the same content as the content image $\mathbf{I}_\mathrm{C}$, then the difference between deeper levels of an encoder V in the feature representation of those two images must be minimised. For this reason, the content loss is simply the mean squared error (MSE) of the features of the content image and the input image that are passed through the encoder V to the layer l_C [1]:

$$\mathrm{L_C}(\mathbf{I}, \mathbf{I}_\mathrm{C}) = \overline{\sum}(V^{l_C}(\mathbf{I}) - V^{l_C}(\mathbf{I}_\mathrm{C}))^2. \tag{1}$$

The grand sum of the following tensor divided by the number of elements, i. e. the grand mean, is denoted here by the overlined sum symbol without indices [18].

Various implementations do not always work equally well for different data. This is particularly reflected in the style. In NST, recurring patterns in the style image are analysed. This refers to structures and edges as well as colours and shapes [2]. One of the possible approaches is to compute different feature representation correlations of the encoder V [1]. Computation of the style loss $\mathrm{L_S}$ between the synthesised image $\mathbf{I}$ and the style image $\mathbf{I}_\mathrm{S}$ is thus similar to the content loss with the difference that features are not compared directly. Instead the MSE of a correlation measure g is used. The style loss $\mathrm{L}_{\mathrm{S},l_S}$ for a layer l_S thus results in [1]

$$\mathrm{L}_{\mathrm{S},l_S}(\mathbf{I}, \mathbf{I}_\mathrm{S}) = \overline{\sum}(g\left(V^{l_S}(\mathbf{I})\right) - g\left(V^{l_S}(\mathbf{I}_\mathrm{S})\right))^2, \tag{2}$$

where the total style loss $\mathrm{L_S}$ is the weighted sum of the selected layers $\mathrm{L_S}$ with [1]

$$\mathrm{L_S}(\mathbf{I}, \mathbf{I}_\mathrm{S}) = \sum_{l_S \in L_\mathrm{S}} \lambda_{l_S} \cdot \mathrm{L}_{\mathrm{S},l_S}(\mathbf{I}, \mathbf{I}_\mathrm{S}). \tag{3}$$

Using multiple layers captures style elements of varying detail. Whereby finer details are represented in the shallower layers and coarser details in the deeper layers [1].

GATYS et al. use the GRAM-Matrix as the correlation measure g of the generated feature map. The GRAM-Matrix of a flattened feature map $\mathbf{x}$ of an encoder V is calculated as follows [11]:

$$\text{gram} : \mathbb{R}^{N\times C} \to \mathbb{R}^{C\times C}, \quad \text{gram}(\mathbf{x}) = \mathbf{x}^T \cdot \mathbf{x}. \tag{4}$$

The images to be compared mostly have a different size. In order for the correlation measure g to be independent of this size, the GRAM-Matrix is divided by the number of elements within the spatial dimensions [1]:

$$g : \mathbb{R}^{H\times W\times C} \to \mathbb{R}^{C\times C}, \quad g(\mathbf{X}) = \frac{\text{gram}(\text{spatvec}(\mathbf{X}))}{H \cdot W}. \tag{5}$$

The function $\text{spatvec}(\cdot)$ is a special case of the vectorisation function. This is used to flatten the feature maps of the encoder V by reducing the spatial dimension of the feature maps to one [19].

Equation (1) for content loss and (3) for style loss are defined in the original formulation of GATYS et al. [1] to automatically identify the content and style of images that need to be merged in a final step for style transfer. The loss function L for the NST consists of a weighted sum of these loss functions and is defined as [1]

$$\mathrm{L}(\mathbf{I}, \mathbf{I}_\mathrm{C}, \mathbf{I}_\mathrm{S}) = \lambda_\mathrm{C} \cdot \mathrm{L}_\mathrm{C}(\mathbf{I}, \mathbf{I}_\mathrm{C}) + \lambda_\mathrm{S} \cdot \mathrm{L}_\mathrm{S}(\mathbf{I}, \mathbf{I}_\mathrm{S}). \tag{6}$$

The weights λ_C and λ_S are hyperparameters and adjusted according to user preferences [1].

The final step in NST is the minimisation of the above mentioned loss function by solving an optimisation algorithm. The synthesis of a new image $\mathbf{I}$ is achieved by iteratively adjusting its pixel values as follows [1]:

$$\mathbf{I} = \arg\min_{\mathbf{I}} \mathrm{L}(\mathbf{I}, \mathbf{I}_\mathrm{C}, \mathbf{I}_\mathrm{S}). \tag{7}$$

In summary, the original formulation of NST consists of a procedure in which an image $\mathbf{I}$ to be synthesised is transformed in an optimisation process. For this purpose, a loss function consisting of two components is determined. The content loss function L_C, which specifies how far the image $\mathbf{I}$ is from the content of the content image $\mathbf{I}_C$ and the style loss function L_S, which specifies how far the image $\mathbf{I}$ is from the style of the style image $\mathbf{I}_S$ [1].

3 Related Work

The seminal work of GATYS et al. [1], presented in the last section, describes the creation of artistic images by separating and recombining image content and style. Since the publication of this first NST approach in 2016 numerous improvements and developments have been published. The following overview is not exhaustive — for a more comprehensive technical overview, please refer to the work of JING et al. [21].

A part of NST work has focused on improving the transfer by various loss functions for style representations. There is no uniform definition for the style of an image. However, two different approaches exist to describe the style in NST [1, 22]. A stochastic approach – such as the GRAM-Matrix concept – assumes that if the global statistics match, the underlying style also matches [23]. A fundamentally different approach is that styles are described by regular or irregular compositions of small patches [23]. Such a structural patch-based approach has been proposed by LI and WAND [22] which also works on the features of an encoder V. Other methods focus on improving the results by including additional losses [24, 25] or a combination of stochastic and structural approaches [14, 24]. That such an image-based method can also be used to transfer the intaglio style has been demonstrated by MEIER et al. [14]. This IST algorithm enables the production of high-quality gravure prints for portrait images within a few hours.

The drawback of all these image optimisation procedures is the slow optimisation process described in (7) in which each individual image is stylised. For a more dynamic NST, these methods have been extended using the same loss function by training a feed forward Convolutional Neural Network (CNN) –

the transformer G – to perform the style transfer. The first to publish such a model-based approach were JOHNSON et al. [5] and ULYANOV et al. [6] whose approaches differ only in the structure of the transformer G. They were able to show that such a network can be trained on one style image and then stylise arbitrary content images in this style without optimisation procedure. However, this also means that a separate network must be trained for each style image. Therefore, other Multi Style Transfer (MST) approaches have investigated the possibility of learning several styles at the same time [8, 9, 10].

In CHEN et al. [10] filter banks consisting of several convolutional layers, each encoding one style, are used for the transformation. ZHANG and DANA [9] on the other hand introduce a *CoMatch* layer that matches feature statistics based on the given styles. In these approaches, a training process is still required to learn weights that are used for transformation. The advantage is that this transformation is adapted to a given style. Nevertheless, there are methods that are based on a more flexible transformation [8, 26, 27]. The basic idea of these approaches is that the statistics of the content features are directly adapted to the style features. DUMOULIN et al. [26] use an adaptive instance normalisation layer to determine the parameters for the transformation. This approach is extended by HUANG and BELONGIE to arbitrary images [27]. An alternative transformation to instance normalisation is a Whitening and Colouring Transformation (WCT) introduced by LI et al. [8], which achieves the transformation by uncorrelating the content features and correlating the features using the style statistics [8]. These approaches have the advantage of being able to perform an universal style transfer with little or no training. The disadvantage is that unlike the previous learnable MST approaches, the transformation cannot be changed by adjusting parameters. This means that if the result is different than expected, a different transformation must be used to change the result.

The images used consist of regions that correspond to different foreground objects and backgrounds or other segmentations. Often stylistic artefacts can arise which destroy the image content [11]. For this reason, spatial control is often integrated into NST to learn better semantic style representation. The feasibility of such spatial control for the reduction of stylistic artefacts has been demonstrated by CHAMPANDARD [12]. An approach that GATYS et al. [11]

have also used for a GRAM-Matrix-based optimisation. These works focusing on semantic style transfer have often been developed for the image optimisation process, but can also be applied to model-based NST approaches. HUANG and BELONGIE [27] show localised stylisation control in their network, while LI et al. [8] use the style/content relationship to control their feedforward networks. Other methods, on the other hand, only integrate spatial control after the training [10, 28]. This makes it possible to stylise the individual regions in different styles, but the actual sense of spatial control during training is lost.

Nevertheless, the drawback of the semantic style transfer is that it has been developed mainly for the optimisation techniques on a single image. If MST models are used, then these models integrate the segmentation only after the training. This means that the models are pre-trained without semantic distinction and can subsequently transfer only the globally learned style representations. The application of these global style is not suitable for the IST, as the intaglio style depends strongly on the respective regions. This means that the intaglio style in the eye, for example, is very different from the style in other regions. For this reason, in MEIER et al. the result could be improved by spatial control. However, such a control has not yet been implemented with the existing MST procedures. The next section therefore explains how existing methods can be modified to integrate a semantic style transfer into the MST.

4 Approach

This section is divided into three parts. First, the procedure of semantic style transfer in the optimisation techniques and the basic functioning of an MST transformer are explained. Subsequently, an approach is proposed with which a semantic control is integrated into the model-based NST to restrict the transformation to explicit regions. Finally, the architecture implemented in this paper based on an existing MST approach is presented.

4.1 Explicit Semantic Model-Based Style Transfer

The semantic control in NST approaches is achieved by segmenting out individual regions in the loss calculation. For this purpose, GATYS et al. [11] use spatially guided GRAM-Matrices for each region $r \in R$ in the synthesised image $\mathbf{I}$ and style image $\mathbf{I}_\mathrm{S}$, which are calculated by multiplying the feature maps by a mask $\mathbf{M}_r^{l_S}$. The corresponding GRAM-based regional style loss $\mathrm{L}_{\mathrm{S},l_S}$ can be formally calculated with [11]:

$$\mathrm{L}_{\mathrm{S},l_S}(\mathbf{I},\mathbf{I}_\mathrm{S}) = \overline{\sum} \sum_r^R \left(g\left(\mathbf{M}_{C,r}^{l_S} \circ V^{l_S}(\mathbf{I})\right) - g\left(\mathbf{M}_{S,r}^{l_S} \circ V^{l_S}(\mathbf{I}_\mathrm{S})\right)\right)^2. \tag{8}$$

The $\circ$ represents an element-wise multiplication with each feature map. The GRAM-Matrix is calculated as before in (4), but is normalised by the area A_r of the mask $\mathbf{M}_r$ to reduce size differences in the masks. An adjusted formula for the GRAM-Matrix in (5) results in [11]:

$$g: \mathbb{R}^{H\times W\times C} \to \mathbb{R}^{C\times C}, \quad g(\mathbf{I}) = \frac{\mathrm{gram}(\mathrm{spatvec}(\mathbf{M_r} \circ \mathbf{I}))}{A_r}. \tag{9}$$

Semantic style transfer is thus achieved by masking individual areas. This makes it possible to restrict the transfer of style to user-defined regions in both the content image and the style image. An example of an improvement through the spatial control can be seen in Figure 2. Such a semantic distinction can also be used for the model-based MST approaches to restrict the transformation to masked regions. In order to propose a possible approach for this distinction, the general synthesis of an MST is first defined.

The synthesis with a model-based NST or transformer G architecture is shown in Figure 3. Conceptually, the architecture consists of three components [9]:

1. The *encoder part* E is applied for feature extraction and dimension reduction to perform transformation on features.
2. The *transformation part* T is used to transform the features in a style.
3. The *decoder part* D consists of convolutional layers that enlarge the input and recreates an output image from the transformed features.

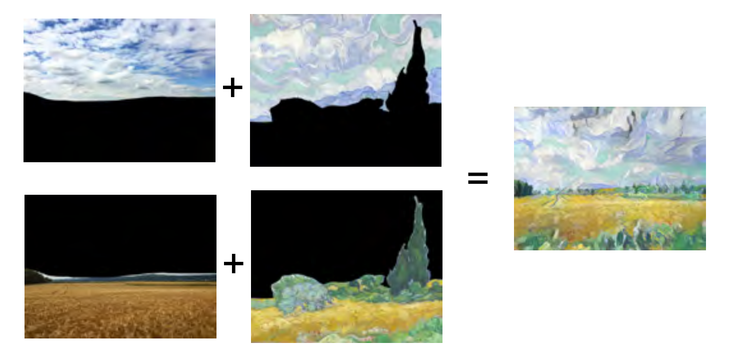


Figure 2: Example of the semantic NST with an explicit transfer for the region *sky* and region *wheatfield*. It can be seen that the result of the semantic style transfer has fewer stylistic artefacts compared to Figure 1, such as the green artefacts in the clouds.

A general synthesis of the transformer G for a content image $\mathbf{I}_\mathrm{C}$ and a style image $\mathbf{I}_\mathrm{S}$ can thus be formulated as:

$$G(\mathbf{I}_\mathrm{C}, \mathbf{I}_\mathrm{S}) = \mathrm{D}(\mathrm{T}(\mathrm{E}(\mathbf{I}_\mathrm{C}), \mathrm{E}(\mathbf{I}_\mathrm{S}))). \quad (10)$$

For a semantic style transfer based on the transformer G in MST, the masks are normally integrated after the training in order to be able to mask the respective areas and to perform the selected transformation T_i in the individual areas. A synthesis with two different style images $\mathbf{I}_{\mathrm{S},1}$ and $\mathbf{I}_{\mathrm{S},2}$ on the basis of a MST transformer G can thus be carried out as:

$$G(\mathbf{I}_\mathrm{C}, \mathbf{I}_{\mathrm{S},1}, \mathbf{I}_{\mathrm{S},2}, \mathbf{M}_C) = \mathrm{D}\left(\sum_{i=1}^{2} \mathrm{T}_i(\mathbf{M}_{i,C} \circ \mathrm{E}(\mathbf{I}_\mathrm{C}), \mathrm{E}(\mathbf{I}_{\mathrm{S,i}}))\right). \quad (11)$$

The transformation T_i has been trained to perform stylisation for the corresponding style images $\mathbf{I}_{\mathrm{S},1}$ and $\mathbf{I}_{\mathrm{S},2}$. This makes it possible to stylise the areas masked with the content masks $\mathbf{M}_C$ in the different styles. However, different sub-styles in a single style image cannot be taken into account in this way.

Therefore, this contribution proposes the implementation of explicit semantic mapping of the transformation. This means that, similar to the spatial control in semantic style transfer, the transformation $\mathrm{T}_{\mathbb{S},r}$ is performed using masks

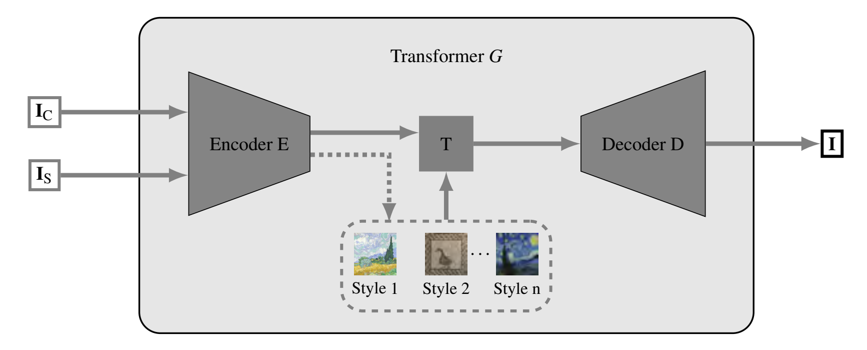


Figure 3: Conceptual structure of an MST transformer G. The structure consists of the encoder part E, transformation part T, and the decoder part D. As an example, three style images are shown for which a stylisation can be carried out with this MST transformer.

for regions $r \in R$ with a certain label. The schematic structure of the explicit semantic transformer is shown in Figure 4. An extended formula for (10) of an explicit semantic transformer $G_\mathbb{S}$ is given by

$$G_\mathbb{S}(\mathbf{I}_\mathrm{C}, \mathbf{I}_\mathrm{S}, \mathbf{M}_C, \mathbf{M}_S) = \mathrm{D}\left(\sum_{r}^{R} \mathrm{T}_{\mathbb{S},r}\left(\mathbf{M}^\mathrm{E}_{C,r} \circ \mathrm{E}(\mathbf{I}_\mathrm{C}), \mathbf{M}^\mathrm{E}_{S,r} \circ \mathrm{E}(\mathbf{I}_\mathrm{S})\right)\right). \tag{12}$$

Thus, in addition to the content image $\mathbf{I}_\mathrm{C}$ and the style image $\mathbf{I}_\mathrm{S}$, the semantic transformer $G_\mathbb{S}$ also receives the masks $\mathbf{M}$ for all regions $r \in R$ of the content and style images. This masks are used to explicitly distinguish the transformation in the feature space. Or in other words, the transformation $\mathrm{T}_{\mathbb{S},r}$ is performed only for regions with the same label in the content and style image. The fusion of the individual regions can be accomplished by a simple sum while maintaining the same dimension.

This label-based distinction of regions and the application of the transformation only to the masked features of the encoder E results in the explicit semantic style transfer. Furthermore, the transformer will only be able to learn the local style representations of the sub-style.

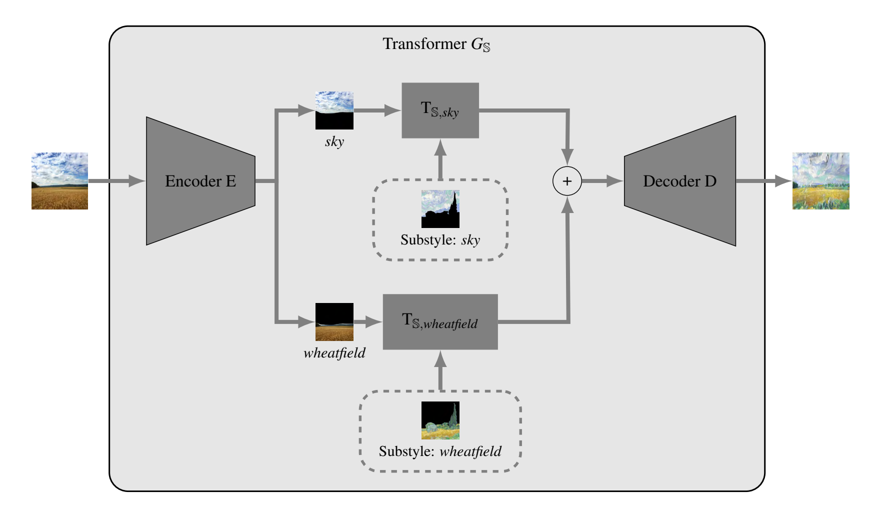


Figure 4: Conceptual structure of a semantic transformer $G_{\mathbb{S}}$. The structure consists of the components encoder E, decoder D and the semantic transformation parts $T_{\mathbb{S},r}$ for the regions $r \in \{sky, wheatfield\}$. The transformation parts are trained to transfer the respective masked sub-styles of the style image $\mathbf{I}_S$ by Vincent van Gogh, "*Wheat Field with Cypresses*".

4.2 Transformer Architecture

The approach presented in the last section can be implemented on the basis of existing methods by adding the semantic transformation part. In this paper, the generative multi-style network (MSG-Net) architecture of ZHANG and DANA [9] is applied. The reason for this selection is that it can be shown that the semantic approach can be integrated into existing approaches. On the other hand, this approach can be learned. Therefore, the transformation does not have to be selected or adapted for each sub-style as in universal style transfer [8, 27].

ZHANG and DANA use a residual block architecture for the MST that can learn the transformation for multiple styles simultaneously [9]. Each residual block contains a branch that leads to a series of convolutional blocks whose outputs are added to the input $\mathbf{X}$ of the block. For a reduction (downsampling) and increase (upsampling) of the input dimension, additional convolutional layers

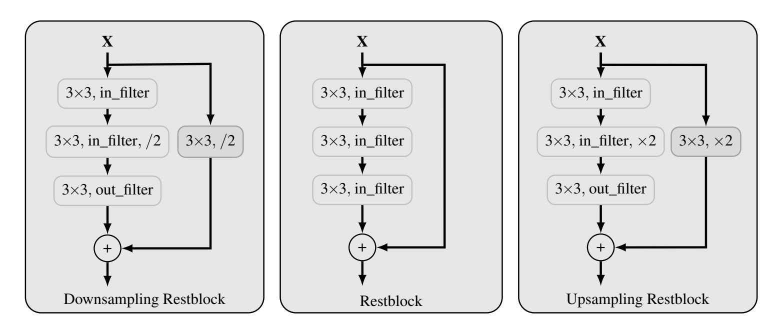


Figure 5: Schematic representation of the residual blocks used in this work for the transformer architecture. The downsampling residual block reduces the input dimension by half, the residual block maintains the input dimension, and the upsampling residual block increases the input dimension by twice. (Based on Figure 5 in [9]).

are used in the direct branch in MSG-Net. A general representation of the residual blocks used in this work is shown in Figure 5.

This architecture allows the layers to learn modifications of the identity mapping rather than the entire transformation, which has been shown to be beneficial for deep neural networks [29]. Each residual block consists of several convolutional blocks. These convolution blocks consist of the following three components [29]:

1. an instance normalisation layer [25],
2. a Rectified Linear Unit (ReLU) Activation Function [9], and
3. a convolutional layer.

For the intaglio style, an adjustment must also be made. The reason for this is that this style is binary, i.e. black or white. Therefore, for simplicity, the intaglio style is treated as greyscale-image in this work. For this purpose, the transformer is adapted accordingly by setting the input and output dimension of the transformer to one, in contrast to the RGB images with three dimensions.

4.2.1 Encoder and Decoder

The encoder E and the decoder D are symmetrically constructed. The encoder consists of an input convolutional layer with a kernel size of 7×7 and 32 filters, followed by two downsampling residual blocks. The decoder consists of two upsampling residual blocks and one output convolutional block with a kernel size of 7×7 [9].

4.2.2 Transformation

The transformation part T is the core element of MSG-Net. The transformation is achieved with the help of a *CoMatch* layer and several residual blocks [9]. In this layer, the GRAM-Matrix of the style is computed at runtime. A weight matrix ω is used to adjust the content features based on the style. The transformation of such a *CoMatch* layer can be defined as [9]:

$$\mathrm{T}(\mathbf{I}_\mathrm{C}, \mathbf{I}_\mathrm{S}) = \mathrm{spatvec}^{-1}\Big(\mathrm{spatvec}(\mathrm{E}(\mathbf{I}_\mathrm{C}))^T \omega g(\mathrm{E}(\mathbf{I}_\mathrm{S}))\Big)^T.$$

For the semantic transformer $G_\mathbb{S}$, a semantic transformation $\mathrm{T}_{\mathbb{S},r}$ is integrated for each region $r \in R$. For this purpose, the features from the encoder E are masked from both the style image $\mathbf{I}_\mathrm{S}$ and the content image $\mathbf{I}_\mathrm{C}$ by the respective masks $\mathbf{M}_{C,r}$ and $\mathbf{M}_{S,r}$. This results in the transformation $\mathrm{T}_{\mathbb{S},r}$ learning and applying only local style representations of the sub-style in region r. A semantic transformation with an *CoMatch* layer can thus be defined as:

$$\mathrm{T}_{\mathbb{S},r}(\mathbf{I}_\mathrm{C}, \mathbf{I}_\mathrm{S}) = \mathrm{spatvec}^{-1}\Big(\mathrm{spatvec}(\mathbf{M}_{C,r} \circ \mathrm{E}(\mathbf{I}_\mathrm{C}))^T \omega g(\mathrm{E}(\mathbf{M}_{S,r} \circ \mathbf{I}_\mathrm{S}))\Big)^T.$$

Each transformation $\mathrm{T}_{\mathbb{S},r}$ consists of a *CoMatch* layer followed by three residual blocks with 128 channel. Furthermore, the *CoMatch* layer is adapted to the use of masks by normalising the GRAM-Matrix by the area A_r of the mask, as shown in (9). This serves to reduce intensity differences due to different mask sizes and thus intensity artefacts [11].

5 Evaluation

This section is divided into two subsections. At first, the dataset and the methods utilised throughout this section are described[1]. Secondly, the results of the trained semantic transformer are shown. Since there is no objective measure of comparison for the evaluation, this part is only of qualitative nature.

5.1 Dataset

The dataset *CelebAMask-HQ* [30] contains 30,000 portrait images from the larger portrait dataset *CelebA* [31]. For each of these images, there are up to 19 of labelled masks for all facial components and accessories, such as *hair*, *eyes*, *earring* and *cloth*, which are additionally split into left and right side of the face [30].

For semantic transfer with a transformer $G_{\mathbb{S}}$, such a division into 19 masks is not necessary. Furthermore, the style will not differ significantly when the left or right component is considered [32]. For this reason, the number of masks is reduced for this work. In Table 1 this classification is shown with the corresponding segmentation masks from the *CelebAMask-HQ* dataset. This twelve mutually exclusive binary masks are used for the training, whereby individual regions, such as *'headwear'*, *'accessories'* or *'glasses'* are optional. A script for the conversion of the masks is available in the implementation.

5.2 Methodology

The implementation of the described transformer architecture has been implemented within the *PyTorch* framework [33], using the NST library *pystiche* [34]. This library allows to implement approaches for NST without much prior knowledge about NST and Machine Learning (ML).

[1] The source code to reproduce the results is published under `https://github.com/jbueltemeier/masterthesis_bueltemeier`

Table 1: Listing of the reduced number of segmentations and the corresponding labels from the *CelebAMask-HQ* dataset.

Training segmentation label	*CelebAMask-HQ* segmentation label
'background'	'background'[1]
'skin'	'skin', 'neck'
'nose'	'nose'
'glasses'	'eye_g'
'eye'	'l_eye', 'r_eye'
'brows'	'l_brow', 'r_brow'
'ears'	'l_ear', 'r_ear'
'lips'	'mouth', 'u_lip', 'l_lip'
'hair'	'hair'
'headwear'	'hat'
'accessoire'	'ear_r', 'neck_l'
'body'	'cloth'

[1] Corresponds to the image pixel that is not covered by the segmentation masks in *CelebAMask-HQ*.

An implementation of the MSG-Net model training is implemented once with the proposed semantic differentiation in the transformer and once without. These two procedures can be described as follows:

1. E $\rightarrow$ T $\rightarrow$ D transformer G and
2. E $\rightarrow$ $T_{\mathbb{S}}$ $\rightarrow$ D semantic transformer $G_{\mathbb{S}}$.

The first procedure without semantic control only involves one transformation for the style image used. The second method performs the transformation of the transformer using the approach proposed in (12), whereby the transformation is performed for all possible areas $r \in R$.

The intaglio motifs required for the style images have been cut out from high-resolution scans of banknotes and converted into greyscale. The scans were provided by *Koeing & Bauer Banknote Solutions*. The style images used in this work are shown in Figure 9 in the appendix. Each image is resized within the *pystiche* library with a bilinear interpolation to a size of 512 on the short side.



Substyle *body*



Substyle *hair*



Substyle *skin*

Figure 6: Different sections of the *UAH020 S1997* Banknote are shown (Courtesy of *Koeing & Bauer Banknote Solutions*). There are clear differences in the shapes used between the individual images, each representing its own sub-style.

The transformers are trained with a batch size of 1 for 30000 iterations and the optimisation is performed by the Adam algorithm [35] with a learning rate of 10^{-4} [33]. For the calculation of the content loss and the style loss, the 19-layer VGG network [15] is adapted for the feature extraction of the encoder E. The weights and the necessary preprocessing from the *Caffe* framework[36] are used for this encoder. The style loss of the transformer G is calculated with (2) and the transformer $G_{\mathbb{S}}$ with (8). The feature extraction is performed in shallower layers of the encoder V than in the original papers because the intaglio structures are very fine. The layer $l_C = \texttt{relu_2_2}$ has been used for content loss and the layers $l_S \in L_S = \{\texttt{relu_1_1}, \texttt{relu_2_1}, \texttt{relu_3_1}\}$ for the style loss. The weights of the individual layers in the style loss are calculated by $\lambda_{l_S} = 1/n_{l_S}^2$, where n_{l_S} denotes the number of channels on the layer l_S [11]. The hyperparameters for the weighting of the loss functions from (6) have been chosen empirically. They correspond to $\lambda_C = 1$ for the content loss and $\lambda_S = 10^1$ for the style loss of the transformer G and the transformer $G_{\mathbb{S}}$.

5.3 Intaglio Style Transfer

The attention to detail and the different ways in which the engraver can engrave the print design distinguish the intaglio style [37]. Typical features of engraving are lines and shading by cross-hatching, where overlapping lines create darker areas [37]. These possibilities lead to differences in style. This is illustrated in Figure 6.

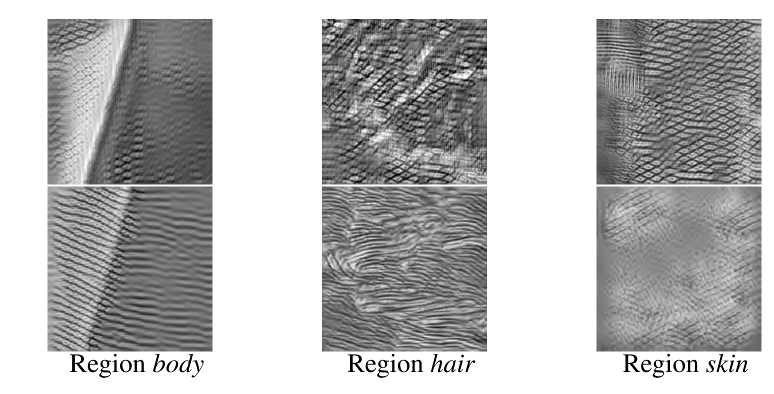


Figure 7: Style transfer of a transformer G (top) and transformer $G_{\mathbb{S}}$ (bottom) for the regions *body*, *hair* and *skin* of a portrait image.

The fact that a transformer G is not capable of learning such a distinction is particularly evident in a detailed view. For this purpose, stylised image sections for the regions *hair*, *skin* and *body* can be seen in the Figure 7. It can be seen that the results of the transformer G do not differ in all three areas. In contrast, the results of the semantic transformer $G_{\mathbb{S}}$ show that these transformers are able to learn and transfer a semantic distinction of the sub-styles. The difference is particularly noticeable in the *hair* sub-style with the wavy lines compared to the diamond pattern in the other two detail images.

These differences in the sub-styles in the regions can also be found in the stylisation of complete portraits. Figure 8 shows an example of this. To emphasise the intaglio structures in the portrait, the background in the result images is displayed uniformly in white. Overall, it shows that both the transformer G and the semantic transformer $G_{\mathbb{S}}$ can be used to create high-quality intaglio images. However, the transfer of the different sub-styles in the semantic transformer $G_{\mathbb{S}}$ lead to an improvement of the results in terms of the intaglio style.

Nevertheless, a disadvantage of the semantic transformer $G_{\mathbb{S}}$ is that transition areas between the regions arise. This effect can be seen especially at the hairline in the result image of the semantic transformer $G_{\mathbb{S}}$ in Figure 8. This makes this result appear qualitatively inferior compared to the transformer G. To reduce this effect, it makes sense to create a binary edge between the

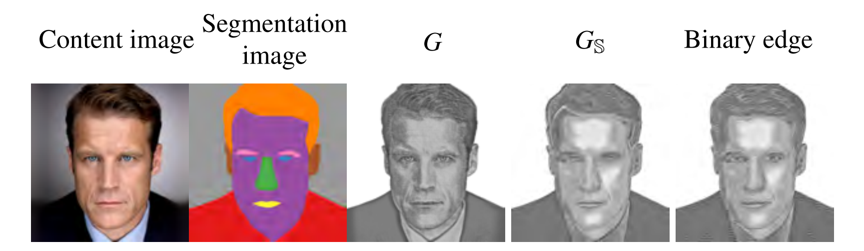


Figure 8: Result of a stylisation of a complete content image (left) with the segmentation used for this, which marks the different regions in colour. From left to right, the result of a transformer G, a semantic transformer $G_{\mathbb{S}}$, and the result with a binary edge between the regions is shown.

individual regions. The reason for this is that with very different sub-styles, no transitional area is expected anyway. This binary edge can be achieved by the semantic transformer $G_{\mathbb{S}}$ stylising the respective regions separately one after the other and merging them in pixel space. The result of such a stylisation is shown in Figure 8 on the right. The result shows that the transition areas can be reduced and the overall result is better.

6 Conclusion and Outlook

This paper has proposed an approach for a semantic treatment of sub-styles in a model-based NST. It is shown that the semantic treatment of a style image leads to an improvement of the result. This is because with a semantic transformer $G_{\mathbb{S}}$, in contrast to a transformer G without semantic transfer, it is possible to transfer the different sub-styles to certain regions in the content image. The semantic transformer is thus able to produce high quality intaglio images.

However, there are still questions to be answered in future work:

1. An assumption for simplification is that the intaglio image is greyscale. This means that there are transitions between black and white. This is not the case in practice. For this reason, a binary post-processing must take place in order not to complicate the machine readability of the Intaglio style.

2. It has been shown that with the intaglio style, edge artefacts occur between different regions. Therefore, merging the regions at pixel space has been investigated to improve the results. These regions should be further investigated. In this context, it is also important to check for which regions masking is necessary at all. However, there is still a lack of a suitable comparison measure.

3. In addition, a single style image has initially been taught for each region. With the implemented approach, it is also possible to learn several sub-styles for a region at the same time. Learning several sub-styles can be used to improve the overall result. For example, it is helpful to use a style image with long hair for long hair and one with short hair for short hair.

Acknowledgment

This contribution is part of the project *Fused Security Features*, which is funded by the *Ministry for Culture and Science of North Rhine-Westphalia* (MKW NRW) under the Grant ID 005-1703-0013. The authors thank the *Koeing & Bauer Banknote Solutions* for supplying high-resolution scans of various banknotes that were utilised as style images for the IST.

7 Template Images

References

[1] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. Image style transfer using convolutional neural networks. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[2] Andrew Glassner. Deep Learning: A Visual Approach. *No Starch Press*, 2021.



Figure 9: Overview over the Intaglio portraits and corresponding segmentation, which are used as style images within the IST. From left to right and top to bottom, the portraits were extracted from the following banknotes: *HUF2000 S2007*, *GBP005 S2002*, *LRD50 S2008*, *MAD050 S2002*, *UAH020 S1997* and *Specimen02*.

[3] Turn Photos From Simple To Sublime With The Style Transfer Tool. *Picsart*, 2022. Online. Retrieved: 15/08/2022. `https://picsart.com/style-transfer`.

[4] Piotr Bojanowski, David Lopez-Paz, Hervé Jegou, and Antoine Bordes. Using AI for new visual storytelling techniques in VR. *Meta*, 2017. Online. Retrieved: 15/08/2022. `https://engineering.fb.com/2017/07/26/virtual-reality/using-ai-for-new-visual-storytelling-techniques-in-vr/`.

[5] Justin Johnson, Alexandre Alahi, and Li Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In: *European Conference on Computer Vision (ECCV)*, 2016.

[6] Dmitry Ulyanov, Vadim Lebedev, Andrea Vedaldi, and Victor S. Lempitsky. Texture networks: Feed-forward synthesis of textures and stylized images. *Computing Research Repository (CoRR)*, 2016.

[7] Artsiom Sanakoyeu, Dmytro Kotovenko, Sabine Lang, and Björn Ommer. A style-aware content loss for real-time hd style transfer. In: *European Conference on Computer Vision (ECCV)*, 2018.

[8] Yijun Li, Chen Fang, Jimei Yang, Zhaowen Wang, Xin Lu, and Ming-Hsuan Yang. Universal Style Transfer via Feature Transforms. *Neural Information Processing Systems (NIPS)*, 2017.

[9] Hang Zhang and Kristin Dana. Multi-style Generative Network for Real-time Transfer. *ArXiv*: `abs/1703.06953`, 2018.

[10] Dongdong Chen, Lu Yuan, Jing Liao, Nenghai Yu, and Gang Hua. StyleBank: An Explicit Representation for Neural Image Style Transfer. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[11] Leon A. Gatys, Alexander S. Ecker, Matthias Bethge, Aaron Hertzmann, and Eli Shechtman. Controlling perceptual factors in neural style transfer. *Computing Research Repository (CoRR)*, 2017.

[12] Alex J. Champandard. Semantic Style Transfer and Turning Two-Bit Doodles into Fine Artworks. *ArXiv*: `abs/1603.01768`, 2016.

[13] Rudolph L. van Renesse. *Optical Document Security*. Artech House, 3. edition, 2005.

[14] Philip Meier, Julian Bültemeier, Volker Lohweg, Helene Dörksen, and Johannes Schaede. Intaglio Style Transfer – Partially Automating the Intaglio Image Creation Process. *Conference on Optical Document Security (ODS)*, 2020.

[15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *Computing Research Repository (CoRR)*, 2014.

[16] Alexei A. Efros and Thomas K. Leung,. Texture synthesis by non-parametric sampling. In: *IEEE International Conference on Computer Vision (ICCV)*, 1999.

[17] Javier Portilla and Eero P. Simoncelli. A Parametric Texture Model Based on Joint Statistics of Complex Wavelet Coefficients. *International Journal of Computer Vision (IJCV)*, 2000.

[18] Brian S. Everitt and Anders Skrondal. *The Cambridge dictionary of statistics*, 2010.

[19] Leslie Hogben. Handbook of Linear Algebra *Chapman & Hall / CRC*, 2007.

[20] Ethem Alpaydin. Introduction to Machine Learning. 4th Edition, *MIT Press*, 2020.

[21] Yongcheng Jing, Yezhou Yang, Zunlei Feng, Jingwen Ye, Yizhou Yu, and Mingli Song. Neural style transfer: A review. *IEEE Transactions on Visualization and Computer Graphics*, 2019.

[22] Chuan Li and Michael Wand. Precomputed real-time texture synthesis with markovian generative adversarial networks. In: *European Conference on Computer Vision (ECCV)*, 2016.

[23] Dongxiao Zhou. *Texture analysis and synthesis using a generic Markov-Gibbs image model.* PhD thesis, University of Auckland, 2006.

[24] Eric Risser, Pierre Wilmot, and Connelly Barnes. Stable and controllable neural texture synthesis and style transfer using histogram losses. *Computing Research Repository (CoRR)*, 2017.

[25] Dmitry Ulyanov, Andrea Vedaldi, and Victor S. Lempitsky. Instance Normalization: The Missing Ingredient for Fast Stylization. *ArXiv*: `abs/1607.08022`, 2016.

[26] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. A Learned Representation For Artistic Style. *ArXiv*: `abs/1610.07629`, 2016.

[27] Xun Huang and Serge J. Belongie. Arbitrary Style Transfer in Real-Time with Adaptive Instance Normalization. *IEEE International Conference on Computer Vision (ICCV)*, 2017.

[28] Zixuan Huang, Jinghuai Zhang, and Jing Liao. Style Mixer: Semantic-aware Multi-Style Transfer Network. *Computer Graphics Forum*, 2019.

[29] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[30] Cheng-Han Lee, Ziwei Liu, Lingyun Wu, and Ping Luo. Maskgan: Towards diverse and interactive facial image manipulation. *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[31] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep Learning Face Attributes in the Wild. In: *International Conference on Computer Vision (ICCV)*, 2015.

[32] Ahmed A. S. Seleim, Mohamed A. Elgharib, and Linda Doyle. Painting style transfer for head portraits using convolutional neural networks. *ACM Transactions on Graphics (TOG)*, 2016.

[33] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In: *Proceedings of the NIPS Autodiff Workshop*, 2017.

[34] Philip Meier and Volker Lohweg. pystiche: A Framework for Neural Style Transfer. *Journal of Open Source Software (JOSS)* `10.21105/joss.02761`, 2020.

[35] Diederik P. Kingma and Jimmy Lei Ba. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR)*, 2015.

[36] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *Computing Research Repository (CoRR)*, 2014.

[37] Adrianus C. J. Stijnman. A history of engraving and etching techniques: developments of manual intaglio printmaking processes, 1400-2000 *Archetype Publications*, London 2012. Online `http://hdl.handle.net/11245/1.378167`.

Determining Feature Importance in Self-Enforcing Networks to achieve Explainable AI (xAI)

Anneliesa Greisbach[1], Christina Klüver[2]

[1]Universität Duisburg-Essen
E-Mail: anneliesa.greisbach@stud.uni-due.de

[2]Universität Duisburg-Essen
Universitätsstr. 12, 45117 Essen
E-Mail: christina.kluever@uni-due.de

1 Introduction

Due to the increasing number of Artificial Intelligence (AI) and Machine Learning (ML) methods and their applications there is a growing demand to understand the results of the respective method (e.g. [1] with reference to the demands in industry 4.0; [2]). While an "explanation component" as in expert systems would be desirable, for many methods this is not applicable.

Explainability is a major challenge, and accordingly, various concepts and methods for achieving Explainable AI (xAI) have been developed in recent years [3], which can be categorized in the taxonomy shown in Fig. 1.

In the stage of explainability, a distinction is made between post-hoc and ante-hoc procedures [2]. In a *post-hoc* procedure, an explanation for a solution occurs after training. This is classified as a *model-agnostic* xAI that applies methods to make the learning process understandable without the actual model passing information to the explanation component. One model-agnostic method is to create a second duplicated model that learns the same data and, upon reaching the same final result, provides information about the basic composition and influence of different features [4][5][6].

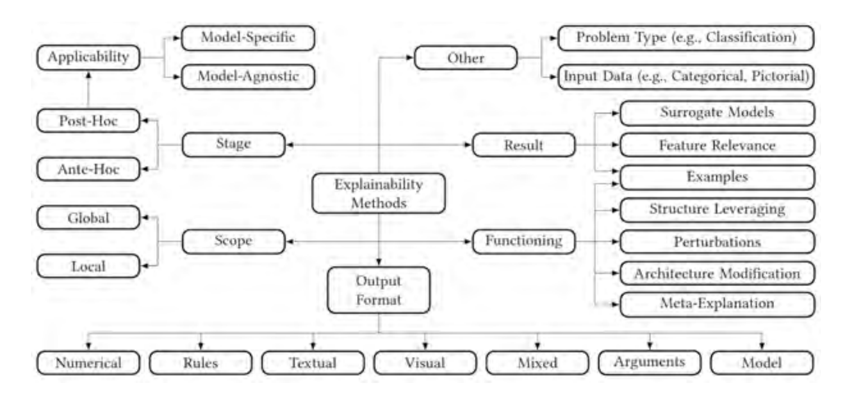


Figure 1: Taxonomy around the topic area of Explainable AI [3, p. 2246]

If the explanation is part of the algorithm, the explainability is designated as *ante-hoc* and is given by design of the algorithm. It is especially suitable for classical machine learning methods, provided that the models are small and manageable. This method is also called *intrinsic xAI*, which is *model specific* and already explainable due to its internal structure. The way of learning can either be transparent (algorithmic transparency), the technical functionality can be clear (simulability), or the algorithm can be decomposed into its individual parts (decomposability) [5][7][8].

Explainable AI algorithms also differ in their *scope*. The scope describes the area to which the explanation refers. It can refer to the functioning of the entire model (global xAI) or to the explanation of the background of a specific decision (local xAI).

In *global xAI*, the explanation component provides an understanding of the general functioning, e.g., the influence of different features on the totality of all decision recommendations. This is used to understand the model and to validate that the features have the expected impact on the decision.

Local xAI provides explanations of a specific decision. This can be used to validate the recommendation or to provide decision support for humans [4][6][9].

The taxonomy of explanatory methods distinguishes whether the outcome of an explanation (*result*) or the operation of an explanatory method is essential (*functioning*). The explanations (*output*) themselves are in turn dependent on the method used and the objective [3].

In this contribution, the meaning of xAI follows the definition of Arrieta et al. [5] according to which an xAI can provide details about a model or explanations of its operation that are understandable to a user. An Explainable AI method for self-organized learning, namely *Self-Enforcing Networks* (SEN) is presented. This method can be used to explain and visualize the impact of a feature on the result. We demonstrate the application of this method to a real case, namely for the decision on the direction of operations at Frankfurt Airport, based on weather data. [10].

AI methods in the broadest sense are used in various areas of air traffic management. Explainability for the methods used is considered by Degas et al. [11] to be necessary for a) describing the algorithms or the results (*descriptive xAI*), b) predicting the behavior of an algorithm (*predictive xAI*), or c) detecting errors or an undesirable behavior of an AI method (*prescriptive xAI*). Accordingly, the solution we present is classified in this case to a) by clarifying the result with a visualization.

The organization of the contribution is as follows: In the next sections, Feature Importance, the concept of Shapley Values, as well as the application of Feature Importance at Self-Enforcing Networks are explained in more detail. In section four, the model of a Self-Enforcing Network as a decision support system regarding the mode of operation at Frankfurt Airport is presented. Recommendations by SEN are shown, especially the benefits resulting from xAI.

2 Feature Importance as a concept of Explainable AI

Within the topic area of Explainable AI, various methods and practices exist that increase the explainability of an algorithm to the user. One of these is explaining the impact of a model's features on the outcome. The impact and

relevance of a feature on the overall model can be referred to as *Feature Importance*.

Determining Feature Importance can help the user understand the output of the model and provides him with a basis of information that can be used to analyze and evaluate a concrete decision recommendation. It can also help the developer of the model to validate and optimize the functioning of the entire model.

2.1 The concept of Shapley Values as the basis of Feature Importance

The calculation of the Feature Importance is based on the concept of Shapley Values from the cooperative game theory of Lord Shapley [12], according to which the influence of a player can be computed considering effects by cooperation and individual performance on the game outcome.

The value that a player contributes to the payoff is called the Shapley Value. Shapley Values can be attributed four characterizing properties. They follow the "null player property", are efficient, symmetric, and linear [13].

Null player property. The „null player property“ means that a player with no share in the final outcome will also get no share in the payoff.

Efficiency. The property of efficiency describes that by mapping the actual influence of an actor on the result, conclusions can also be made about this actor.

Symmetry. The property of symmetry indicates that if two players have the same influence on the outcome, they will have the same Shapley Value. Accordingly, a player with a larger contribution to it also has a higher Shapley Value.

Linearity. The property of linearity states that the sum of all Shapley Values makes up the total influence on the outcome.

Only if these four properties are true, a value can be called Shapley Value. The Shapley Value of a player i on the game result can be usually calculated with the following formula [14]:

$$\Phi_i(v) = \sum_{S:i \in S} \frac{|S|!(n-1-|S|)!}{n!}(v(S \cup \{i\}) - v(S)) \tag{1}$$

The overall contribution of a player i to a game outcome is calculated by the sum of all partial influences resulting from coalitions with other players of the player set S, given n players.

The concept of Shapley Values is already applied to artificial intelligence methods. A Shapley Value thus explains the contribution of features on the output of a model and can therefore also be called Feature Importance. This allows a statement to be made about the contribution of a feature in interaction with other features to the final outcome, e.g., a prediction or classification. [13][15][16].

3 Feature Importance at Self-Enforcing Networks

The idea of computing Feature Importance based on the concept of Shapley Values is already used in the context of supervised learning models [15][16][17] and has now been applied to self-organized learning Self-Enforcing Networks (SEN).

The identification of Feature Importance for Self-Enforcing Networks can be categorized as intrinsically explainable Artificial Intelligence due to the structure and operation of SEN, as shall be demonstrated.

3.1 Self-Enforcing Networks

A Self-Enforcing Network (SEN) is a non-supervised and self-organized learning network that acquires and orders knowledge according to cognitive theory learning models [18][19].

SEN consists of three components: the semantic matrix, the actual neural network, and the visualizations [cf. 20].

The semantic matrix is the basis of the learning process. It contains the essential attributes (features) and their degree of membership to an object [18].

As SEN is used as both Machine Learning and Artificial Intelligence method, the data sets are transferred to the semantic matrix according to the problem. In the first case, sensor data, image data, business data, etc. are imported into the semantic matrix, which are learned and ordered according to their similarity.

If SEN is used as a method of AI, frequently so-called *reference types* are formed in addition by experts, based on data containing the specifics of the use case and expert knowledge. This method is used in the following.

The neural network itself is often two-layered and can be designed as a feed-forward, feed-back, or recurrent network depending on the defined connections between attributes and reference types [20].

The data can be normalized in SEN binary in the interval of (0,1) or bipolar in the interval of (-1,1). For each feature, a cue validity factor (cvf) can be set when building the model. The cue validity factor, following the concept of cognitive psychologist Eleanor Rosch [21], influences the strength of an attribute's effect on activation by the network. The cvf can be determined based on the relevance of the attribute for the use case [18].

The peculiarity of SEN is that the weight matrix is not randomly generated but consists of zeros at the beginning.

The Self-Enforcing Rule (SER) i.e. the learning rule used in SEN transforms the values of the semantic matrix v_{sm} into a weight value between object and attribute w_{oa} with the learning rate c [20]:

$$w_{oa} = c * v_{sm} \tag{2}$$

The learning rate describes how forcefully the network adjusts the weight values after each learning process. If the cue validity factor of the respective

attribute cvf_a is to be taken into account, the learning rule is extended as follows:

$$w(t+1) = w(t) + \Delta w \text{ und} \tag{3}$$
$$\Delta w = c * w_{oa} * cvf_a$$

This learning procedure makes it possible to reconstruct the results at any time [cf. 18].

Seven activation functions are available in SEN; for the model shown here, the Enforcing Activation Function (EAF) has been proven to be effective [24]:

$$a_j = \sum_{i=1}^{n} \frac{w_{ij} * a_j}{1 + |w_{ij} * a_j|} \tag{4}$$

SEN transfers the information of the semantic matrix to the network via the learning rule. After the learning process, the data are ordered according to their similarity: The more alike the data are, the closer they are represented to one another in the so-called map visualization. When new data are presented after the learning process, the activation values indicate the strength of the similarity of the input data to the reference types. The higher the final activations are, the more similar the input data are to the reference types. In addition, the Euclidean distances of the input data to the reference types are also displayed, indicating their similarity by the smallest distance.

3.2 Intrinsic Explainability at Self-Enforcing Networks

The individual activation values of the attributes are extracted from SEN to directly obtain the Feature Importance, or Shapley Values, which are subsequently visualized. The retrieved values are the individual activations of the attributes and indicate the final activation of the input vector set to a reference type when added together.

The properties "null player property", efficiency, symmetry and linearity of Shapley Values are given, since on the one hand the individual values are

extracted and on the other hand, the added values constitute the total activation value. Thus, these can be defined and considered as Feature Importance.

This results from the SEN's mode of operation, which is that the weight values directly reflect the importance of the individual attributes in the reference types, and thus the activation values of the individual attributes have the corresponding effects on the final result.

3.3 Visual representation of Feature Importance as Local Explainable AI

The Feature Importance of Self-Enforcing Networks can be displayed visually to provide the end user with easy access to the values. Either the Feature Importance of the entire model (global xAI) or the Feature Importance of a specific decision (local xAI) can be displayed.

In outlining the influences of a specific recommendation by SEN, the representations listed below are suitable.

The absolute Feature Importance of the input dataset and the classified reference type can be displayed side by side. This allows a quick view of the differences in the activation values of the individual features between the input vector and the reference type (Fig. 2).

In SEN, the output is the reference type that has the highest activation, or the smallest distance, with respect to the new input vector. By applying Feature Importance, the activation is calculated for each individual attribute. Afterwards these can be compared.

In this representation, the bars indicate the absolute Feature Importance of the input vector (gray) and reference type (red). The size of the bars describes the influence of the feature on the total activation value. A bar below the x-axis means negative activation, a bar close to the x-axis means weak activation, a high bar means strong activation.

Another possible representation is the difference between input vector and classified reference type (Fig. 3). For this purpose, the difference in Feature

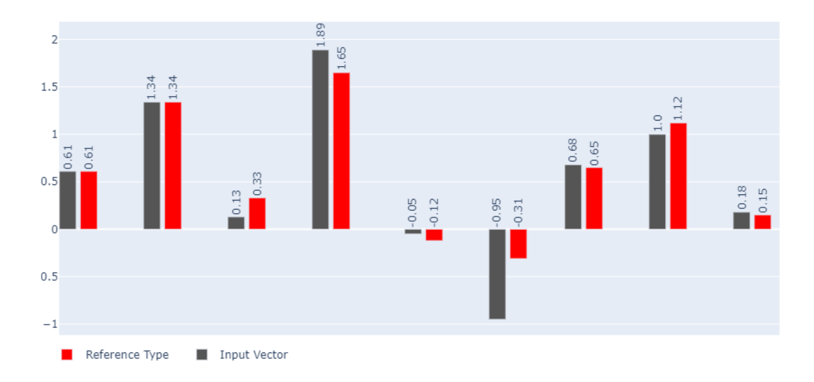


Figure 2: Exemplary absolute feature values of the activation of an input vector and of a reference type by a bipolar normalization.

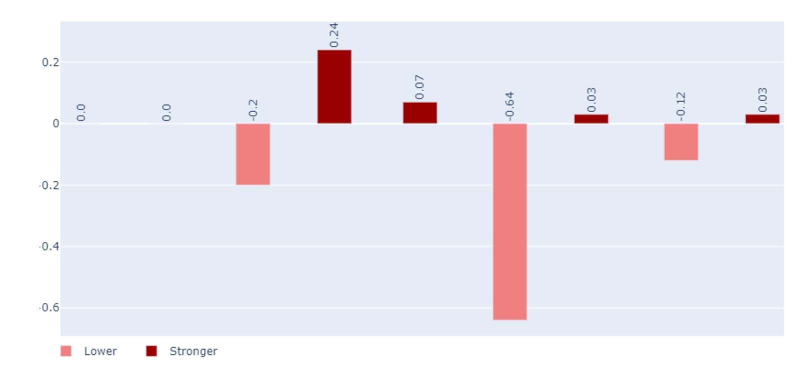


Figure 3: Exemplary difference between input vector and classified reference type.

Importance between the input vector and the classified reference type is calculated and displayed. This representation allows an even easier comparison of the differences and deviations between the Feature Importance of the input vector and the classified reference type.

In this plot, the classified reference type is located on the x-axis. The bars describe the differences between Feature Importance of the input vector and the classified reference type with respect to the individual activation values. A bar below the x-axis means a weaker feature importance, a bar close to the axis

of the classified reference type means a small difference to it, a high bar means a large deviation between input vector and the classified reference type.

In addition to the visualizations presented, further visualizations related to reference types that are not classified but preferred by experts can also be created in case the expert would have made a different decision than the SEN. Both visualizations provide the expert with a basis for decision-making.

4 Use case of the selection of the operating direction at Frankfurt Airport

The determination of the Feature Importance is presented as an example for the recommendation of the operating direction at Frankfurt Airport.

4.1 Description of the use case

Frankfurt Airport has four runways (Fig. 4). Three of them are on a parallel runway system aligned with the compass heading of 250° and 70°, with operating directions designated 25 and 07. Runway West, with a compass heading of 180° and the designation 18, lies nearly orthogonal to the parallel runway system. This runway can only be used for takeoff in one given direction [22].

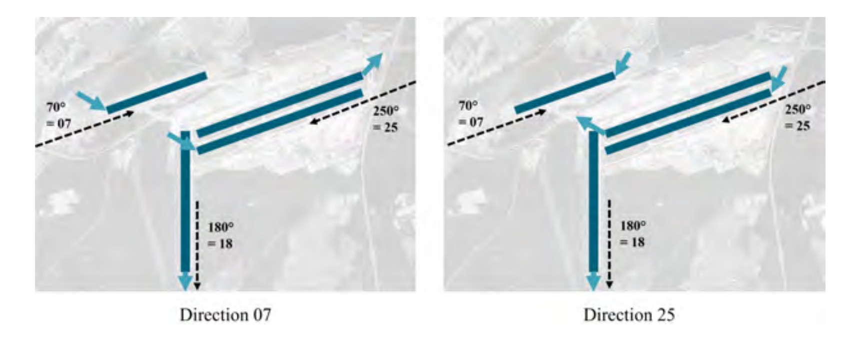


Figure 4: Own representation of the runways at Frankfurt Airport according to [25, p. 376].

The runways on the parallel runway system are each used in the direction specified by air traffic control [23].

The decision on the direction of operations is made by air traffic control on the basis of the current conditions, in particular on the basis of the prevailing wind direction and strength. For this purpose, air traffic control has, among other things, weather forecasts on the wind development, measurements of the wind conditions at the airport near the ground and information from the pilots on the wind conditions during landing [24].

The design of the weather forecast is defined worldwide by the regulations of Annex 3 of the International Civil Aviation Organization (2018). As Terminal Aerodrome Forecast (TAF), the forecasts for wind direction and wind speed, visibility, cloud cover and significant weather phenomena are provided for international airports [24].

Since wind conditions in Central Europe can change rapidly within a short period of time, air traffic control must monitor weather forecasts for the next few hours at all times and continuously review decisions made regarding the direction of the runway [25].

The choice of operating direction is a major challenge for air traffic control because, on the one hand, the safety of crew and passengers must be ensured at all times and, on the other hand, the decision to change the operating direction involves considerable effort and delays, so it must be very well justified [23].

Furthermore, such a decision to change the direction of operations requires a certain lead time in order to be able to coordinate the actual implementation at the airport. Especially in situations with weak winds, the available information is not very meaningful, which complicates the decision of air traffic control [23][24].

The use of a computer-based decision support system can remedy this situation and validate or automate the decision through a second assessment.

In cooperation between the department of aviation meteorology of Deutscher Wetterdienst (the National Weather Service of Germany), which among other things provides the weather forecasts for Frankfurt Airport, and the research

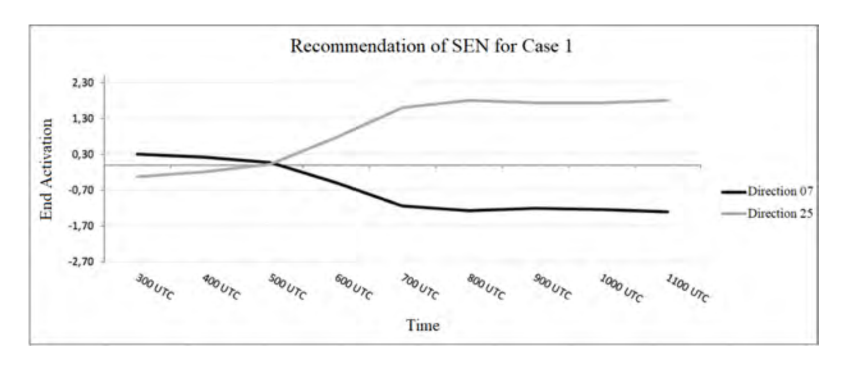


Figure 5: Recommendation on the direction of operation by SEN [23]

group CoBASC, such a decision support system for Frankfurt Airport was developed. The SEN has as parameter settings the Enforcing Activation Function (EAF) a learning rate of 0.5 and 4 learning steps.

Based on the predicted wind conditions, it gives a recommendation for the optimal operating direction to the available weather forecasts. The SEN thus provides a recommendation of whether and when the operating direction should be changed [23].

To simplify and test the first model, the focus was initially placed on the parallel runway system with runway directions 07 and 25. The decision for or against Runway West with runway direction 18 was not initially integrated into the model. To briefly illustrate how SEN works, an example of a recommendation by SEN based on the weather forecast is shown in Fig. 5. At 05:00 UTC, SEN recommends to change the operating direction to direction 25.

Since the recommendations by SEN follow the same trend as those of the controllers, SEN is suitable to serve as a decision support system for Air Traffic Control [23].

4.2 Data of the use case

Air traffic control at Frankfurt Airport uses weather forecasts from the COSMO-DE Ensemble Prediction System (COSMO-DE-EPS), which is provided by the

German Weather Service. As an ensemble forecast model, it provides a total of 20 possible weather forecasts. Uncertainties and probabilities for the forecasts can be expressed by the 20 ensemble members.

For the model in SEN, the path-parallel wind components were calculated at 11 measurement points, meaning the wind strengths and directions aligned with compass degrees 70° and 250°, respectively (Fig. 6). The calculated path-parallel wind component thus provides information on the strength of the headwind and tailwind in knots for aircraft taking off and landing on the glide path. The glide path is the path that aircrafts follow when using the runway.

To represent the uncertainty of the forecasts in SEN, the path-parallel wind component at the 11 measurement points was calculated for all 20 ensemble members. The ensemble members were then used to calculate the quantiles at each of the points with thresholds of 10%, 25%, 50%, 75%, and 90% [24]. Thus, each dataset has five wind components for each of the 11 measurement points, making a total of 55 features in SEN [23]. The cue validity factor was set in SEN per feature as a function of the quantiles and the distance to the runway.

The geographic location of the reference points is shown in Fig. 7.

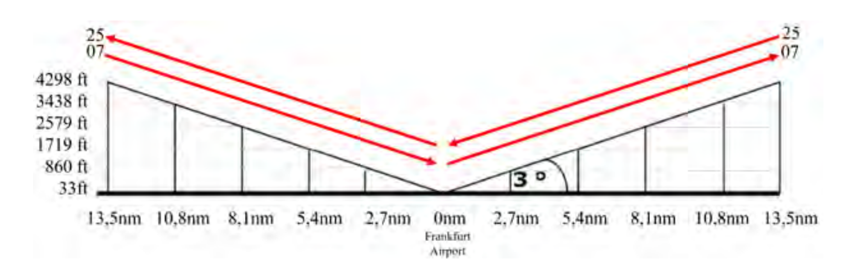


Figure 6: Own model, not to scale, representation of the glide path and the 11 measuring points of runway direction 07 and 25 at Frankfurt Airport based on [24, p. 232].



Figure 7: Geographical location of the measurement points [24, p.232].

4.3 Determination of the Local Feature Importance of the use case

The decision to change the direction of operation involves considerable effort as well as delays and must be well justified. An explanation component is of substantial benefit, especially when wind conditions do not permit an immediate and unambiguous decision.

Since a representation of the 55 features in a visualization would appear unmanageable, it was decided specifically for this use case to add the individual activation values of the five calculated quantiles per measuring point and thus to obtain an activation at the 11 measuring points.

Fig. 8 shows the absolute Feature Importance of reference type 'Direction 07'. The Feature Importance of the measurement points is arranged according to the positioning on the glide path for greater clarity and, when viewed together,

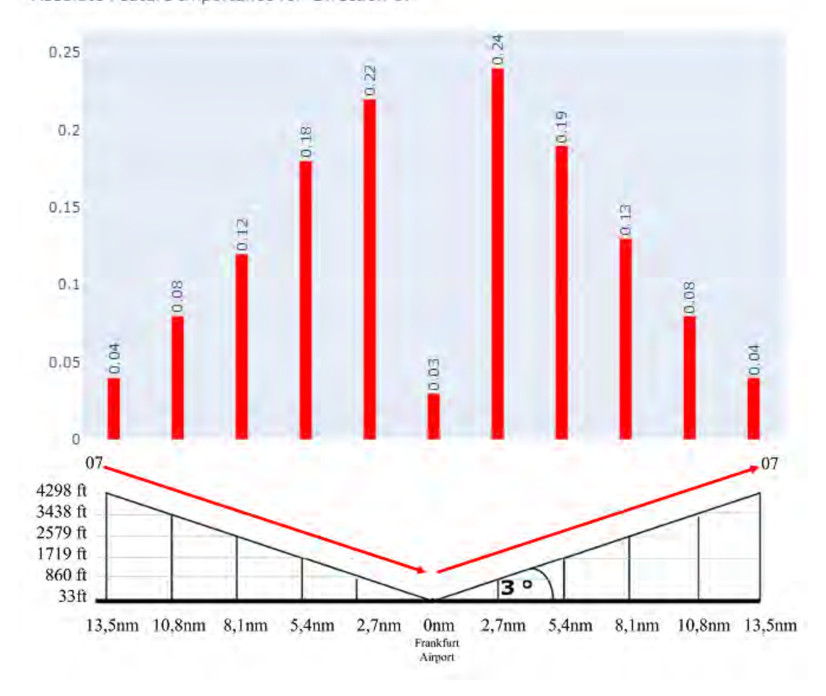


Figure 8: Visualization of the Feature Importance for reference type 'Direction 07' with a model-like, not to scale representation of the measuring points on the glide path at Frankfurt Airport following Zinkhan [24].

allows direct conclusions to be drawn about the wind conditions at the various heights of the glide path.

To visualize the Feature Importance for the use case, the data of "case 1" from [23] is used and analyzed with respect to their Feature Importance. For each of these 9 recommendations between 03:00 UTC and 11:00 UTC, the Feature Importance can be read out to analyze the decision recommendation in more detail. The Feature Importance of the decision at 09:00 UTC is analyzed in depth below.

In Fig. 9 the input vector 09:00 UTC was classified as 'Direction 25'. The Feature Importance with respect to this classification is shown in Fig. 10 as

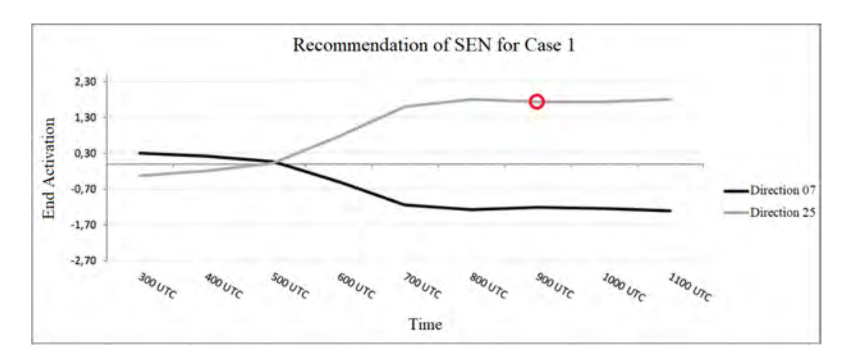


Figure 9: Visualization of the recommendation on the direction of operation by SEN from case 1 at 09:00 UTC [24].



Figure 10: Feature Importance of 09:00 UTC regarding ‘Direction 25’

absolute value and as difference to the Feature Importance of the reference type ’Direction 25’.

Both plots indicate that for input vector 09:00 UTC, the high-altitude winds have a higher activation than those of the reference type. The ground winds, on the other hand, have a lower influence on the final activation.

The final activation of 2.93 of the input vector can be broken down by this visualization and gives experts the opportunity to analyze the classification and the differences to the feature influences on the final result in more detail. Decision support systems, such as this one, thus take on greater meaningfulness.

5 Conclusion and further work

In this contribution, we have shown how Feature Importance can be applied to self-organized learning networks. The architecture and learning procedures of SEN fulfill the properties of Shapley Values. Feature Importance can be read directly by decomposing a vector into its individual components (individual attributes) without additional calculations; thus, SEN can be classified as intrinsic xAI.

The knowledge gained will be integrated into a follow-up project in which weather forecasts are to be improved by using large amounts of data. Since a reference type consists of almost 2,000,000 data points, it is of great importance for decision makers to know which features are decisive for the decision of a runway.

With the developed method, there is no loss of performance in determining Feature Importance, but the number of features must be reduced. One possibility is for the experts themselves to determine which features are relevant to them and should be displayed.

References

[1] I. Ahmed, J. Gwanggil, F. Piccialli. “From artificial intelligence to explainable artificial intelligence in industry 4.0: a survey on what, how, and where”. In: *IEEE Transactions on Industrial Informatics* 18.8. pp. 5031-5042. 2022.

[2] M.R. Islam, M.U. Ahmed, S. Barua, S. Begum. “A Systematic Review of Explainable Artificial Intelligence in Terms of Different Application Domains and Tasks”. In: *Applied Science* 12, 1353. 2022.

[3] T. Speith. “A Review of Taxonomies of Explainable Artificial Intelligence (XAI) Methods”. In: *ACM Conference on Fairness, Accountability, and Transparency.* pp. 2239-225012. 2022.

[4] A. Das, P. Rad: “Opportunities and Challenges in Explainable Artificial Intelligence (XAI): A Survey”. In: *arXiv e-prints (2020): arXiv-2006*. 2020.

[5] A.B. Arrieta, N. Díaz Rodríguez, J. Del Ser, A. Bennetot, S. Tabik, A. Barbado et al. “Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI”. In: *Information Fusion* 58. pp. 82-115 2020.

[6] G. Schwalbe, B. Finzel: “A Comprehensive Taxonomy for Explainable Artificial Intelligence: A Systematic Survey of Surveys on Methods and Concepts” In: *arXiv e-prints, arXiv-2105*. 2021.

[7] R. Guidotti, A. Monreale, S. Ruggieri, F. Turini, F. Giannotti, D. Pedreschi: “A Survey of Methods for Explaining Black Box Models”. In: *ACM Computing Surveys* 51.5. 1019.

[8] B. Mittelstadt, C. Russell, S. Wachter: “Explaining Explanations in AI”. In: *Proceedings of the conference on fairness, accountability, and transparency* pp. 279-288. 2019.

[9] A. Adadi, M. Berrada. “Peeking Inside the Black-Box: A Survey on Explainable Artificial Intelligence (XAI)”. In: *IEEE Access* 6. pp. 52138–52160. 2018.

[10] D. Zinkhan, S. Eiermann, C. Klüver, J. Klüver. “Decision Support Systems for Air Traffic Control with Self-enforcing Networks Based on Weather Forecast and reference types for the Direction of Operation”. In: *Advances in Computational Intelligence. IWANN 2021* (Rojas I., Joya, G., Catala, A. eds.), pp. 404-415. Springer, Cham. 2021.

[11] A. Degas, M.R. Islam, C. Hurter, S. Barua, H. Rahman, M. Poudel, D. Ruscio, M.U. Ahmed, S. Begum, M.A. Rahman, et al. “A Survey on Artificial Intelligence (AI) and eXplainable AI in Air Traffic Management: Current Trends and Development with Future Research Trajectory”. In: *Applied Science* 12. 1295. 2022.

[12] L.S. Shapley "A Value for n-Person Games". In: *Contributions to the Theory of Games (AM-28)* (Kuhn, H.W., Tucker A.W. eds.), 2. pp. 307–318. Princeton: Princeton University Press. 1953.

[13] B. Rozemberczki, L. Watson, P. Bayer, H.-T. Yang, O. Kiss, S. Nilsson, R. Sarkar. "The Shapley Value in Machine Learning". In: *Proceedings of the 31st International Joint Conference on Artifical Intelligence* (L. De Raedt, L., ed.) pp. 5572-5579. IJCAI-ECAI 2022.

[14] L. Bokati, O. Kosheleva, V. Kreinovich, N.N. Thach. "Why Shapley Value and Its Variants Are Useful in Machine Learning (and in Other Applications)". In: *Proc., 15. Workshop Computational Intelligence*. Departmental Technical Reports (CS). 1729. 2022.

[15] S. Lundberg, S.-I. Lee. "An unexpected unity among methods for interpreting model predictions". In: *Proceedings of NIPS 2016 Workshop on Interpretable Machine Learning for Complex Systems* (Wilson, A.G., Kim, B. Herlands, W. eds.) arXiv preprint arXiv:1611.07478 (2016). 2016.

[16] S. Lundberg, S.-I. Lee. "A Unified Approach to Interpreting Model Predictions". In: *Advances in neural information processing systems* 30. pp. 4768-4777. 2017.

[17] H. Chen, S. Lundberg, S.-I. Lee. "Explaining a series of models by propagating Shapley values". In: *Nat Commun* 13. 4512. 2022.

[18] C. Klüver, J. Klüver "KI - Das Self-Enforcing Network (SEN)". In: *Neue Algorithmen für praktische Probleme. Variationen zu Künstlicher Intelligenz und Künstlichem Leben* (Klüver, C. J. Klüver, J. eds.). pp. 9-20. Wiesbaden: Springer Fachmedien. 2021.

[19] C. Klüver, J. Klüver "Chancen und Herausforderungen beim Einsatz neuronaler Netzwerke als Methoden der Künstlichen Intelligenz oder des Maschinellen Lernens in KMU.". In: *Digitalisierung und Nachhaltigkeit – Transformation von Geschäftsmodellen und Unternehmenspraxis. Organisationskompetenz Zukunftsfähigkeit* (M. Bodemann, W.

Fellner, V. Just eds.). pp. 121-148. Berlin, Heidelberg: Springer Gabler. 2022.

[20] C. Klüver, J. Klüver, J. Schmidt. “Modellierung komplexer Prozesse durch naturanaloge Verfahren. Künstliche Intelligenz und Künstliches Leben”. Wiesbaden: Springer Fachmedien. 2021.

[21] E. Rosch. “Natural Categories”. In: *Cognitive Psychology* 4.3. pp. 328-350. 1973.

[22] H. Mensen. “Planung, Anlage und Betrieb von Flugplätzen”. Berlin, Heidelberg: Springer. 2013.

[23] C. Klüver, J. Klüver, D. Zinkhan. “A Self-Enforcing Neural Network as Decision Support System for Air Traffic Control based on probabilistic Weather Forecasts”. In: *International Joint Conference on Neural Networks (IJCNN)*. pp. 729-736. 2017.

[24] D. Zinkhan. “Entscheidungsunterstützungssystem zur Interpretation probabilistischer Wettervorhersagen für den Flughafen Frankfurt”. In: *Neue Algorithmen für praktische Probleme. Variationen zu Künstlicher Intelligenz und Künstlichem Leben* (Klüver, C. J. Klüver, J. eds.). pp. 9-20. Wiesbaden: Springer Fachmedien. 2021.

[25] J.E. Beasley, M. Krishnamoorthy, Y.M. Sharaiha, D. Abramson. “Displacement Problem and Dynamically Scheduling Aircraft Landings”. In: *Journal of the Operational Research Society* 55. pp. 54-64. 2004.

Comparison between Pole Region Design and Model Reference Control for Multivariable TS Fuzzy Systems

Horst Schulte[1]

[1]Control Engineering Group, School of Engineering I, University of Applied Sciences Berlin (HTW Berlin)
Wilhelminenhofstraße 75A, 12459 Berlin, Germany
E-Mail: schulte@htw-berlin.de

Abstract

This paper deals with the comparison of two model-based design methods for multivariable systems in the Takagi-Sugeno form. Model-based means that the multivariable control law is synthesized based on a given mathematical process model and a formal description of the desired control properties. Examined are two methods in which the desired characteristics of the control loop are specified by either a parameterized pole region or a closed-loop reference model, which results in different LMI formulations. In particular, the various LMI-based criteria are discussed, and an illustrative example demonstrates their applicability.

1 Introduction

In general, the systematic design of a model-based controller requires a suitable process model and formal description of the desired closed-loop dynamics. Especially for tasks like multivariable control of nonlinear systems, this paradigm is widely used in the control community. A major part of the model-based design of fuzzy controllers (Takagi-Sugeno form) examines the stabilization

of systems particular stability relaxations and considers less the consideration and guarantee of control performance. This is evident also from the focus of the review papers [3] and [6]. In [3], a systematization of methods for proving global-asymptotic stability (GAS) by Lyapunov inspired verification methods based on common quadratic, piecewise quadratic or fuzzy Lyapunov functions is proposed. An investigation of methods for the consideration of the controller performance is not presented. The overview of the controller design in [6] is also limited to the proof of GAS. Nevertheless, there are approaches to LTI systems that explicitly include a formal description of the desired performance of multivariable control problems. However, these have not yet been sufficiently elaborated for the TS framework. One can distinguish two relevant methods:
First, by applying the LMI region design method of LTI systems [1],[2] to the class of Takagi-Sugeno (TS) fuzzy systems, it is possible to consider the desired control behavior via the parameters of pole regions in the design. Here, a pole region is a subregion of the complex left-half plane which is described by a LMI region [2]. Thanks to the convex combination of the total number of N_r submodels, the transfer of LMI regions to TS systems is straightforward. Instead of $n = 2$, a maximum of $n = N_r^2 + 1$ LMIs must be considered. The last mentioned number is calculated very conservatively without utilizing the submodels' double sum symmetry or common matrices. Note, if the symmetry of the double sum is utilized, then just $n = N_r(N_r + 1)/2 + 1$ are considered in the design. Further details are described in [10].
Second, the desired closed-loop performance of the control can also be specified by a reference model. It is used to describe a desired closed loop input/output characteristic that is directly integrated into the design. The method studied in this paper in the context of TS systems has previously been presented for the class of LPV systems in [4]. Upcoming work will investigate the additional degrees of design freedom provided by TS systems compared to LPV systems. The objective of this paper is first to present the formal design procedures for the fuzzy system framework. The applicability is demonstrated by a mathematical example, which simplifies the dynamics and requirements of power plant units in a coordinated power plant network related to the novel concept of Dynamic Virtual Power Plants (DVPP) proposed in [5].
The paper is structured as follows: Section 2 briefly discusses the LMI formulation for the pole region specification. The controller structure for TS

fuzzy model matching with a reference model and integral state controller as PDC (parallel distributed compensator) structure is presented in Section 3. For illustration, a mathematical example is used in Section 4 that abstractly represents an electrical generation unit in an interconnected power system.

2 TS control design by specification of closed-loop pole region

Given is the control law

$$u = \sum_{i=1}^{N_r} h_i(z) \left(-K_{x,i}\, x + K_{I,i}\, x_I \right) , \quad x_I = \int_0^t \left(y^r(\tau) - y(\tau) \right) d\tau \, , \tag{1}$$

with the introduced auxiliary state vector

$$x_I = \int_0^t \left(y^r(\tau) - y(\tau) \right) d\tau \, , \tag{2}$$

where y^r denotes the external reference value. The control law can be formulated in compact form

$$u = -\sum_{i=1}^{N_r} h_i(z) \underbrace{\left(K_{x,i} \, , \, -K_{I,i} \right)}_{\tilde{K}_i} \tilde{x} \tag{3}$$

by introducing the extended state vector $\tilde{x}^T = \left(x^T \, , \, x_I^T \right)$. In the TS framework the controlled plant is represented by the standard form

$$\dot{x} = \sum_{i=1}^{N_r} h_i(z) \left(A_i x + B_i u \right) , \quad y = \sum_{i=1}^{N_r} h_i(z) C_i x \tag{4}$$

with the convex sum condition $0 \leq h_i \leq 1$ and $\sum_{i=1}^{N_r} h_i(z) = 1 \forall z$. The augmented design model results from (2) and (4)

$$\dot{\tilde{x}} = \sum_{i=1}^{N_r} h_i(z) \underbrace{\begin{pmatrix} A_i & 0 \\ -C_i & 0 \end{pmatrix}}_{\tilde{A}_i} \tilde{x} + \sum_{i=1}^{N_r} h_i(z) \underbrace{\begin{pmatrix} B_i \\ 0 \end{pmatrix}}_{\tilde{B}_i} u + \underbrace{\begin{pmatrix} 0 \\ I \end{pmatrix}}_{\tilde{E}} y^r \,. \tag{5}$$

By substituting u in (5) by the control law (3) follows the augmented closed-loop dynamics

$$\dot{\tilde{x}} = \sum_{i=1}^{N_r} \sum_{j=1}^{N_r} h_i(z) \, h_j(z) \big(\tilde{A}_i - \tilde{B}_i \tilde{K}_j\big) \tilde{x} + \tilde{E} w \,. \tag{6}$$

Note the different indexing of the plant model and the controller due to different input matrices resulting in the double sum. The design for determining the controller gain $\tilde{K}_j$ is based on the specification of a desired pole region $S(\alpha, r, \theta)$ of all eigenvalues λ_{ij} given by

$$\lambda_{ij} = \mathrm{eig}\big(\tilde{A}_i - \tilde{B}_i \tilde{K}_j\big) \quad \text{for} \quad i, j = 1, \ldots, N_r \,. \tag{7}$$

Thereby the pole region is defined by the parameters α,r, and θ as shown in Figure 1. The final design of the controller is based on the following LMI formulation: All eigenvalues λ_{ij} (7) of the closed-loop system (6) with the PDC control law (3) are located in the given pole region $S(\alpha, r, \theta)$ if there exists a common symmetric matrix $X \succ 0$ and $M_j, \; j = 1, 2, \ldots, N_r$ as feasible solution of the LMI problem

$$\begin{aligned}
\Gamma_{ij}^1 &= \tilde{A}_i X + X \tilde{A}_i^T - \tilde{B}_i M_j - M_j^T \tilde{B}_i^T + 2\alpha X \,, \\
\Gamma_{ij}^2 &= \begin{pmatrix} (\tilde{A}_i X + X \tilde{A}_i^T - \tilde{B}_i M_j - M_j^T \tilde{B}_i^T) \sin\theta & (\tilde{A}_i X - X \tilde{A}_i^T - \tilde{B}_i M_j + M_j^T \tilde{B}_i^T) \cos\theta \\ (X \tilde{A}_i^T - \tilde{A}_i X - M_j^T \tilde{B}_i^T + \tilde{B}_i M_j) \cos\theta & (\tilde{A}_i X + X \tilde{A}_i^T - \tilde{B}_i M_j - M_j^T \tilde{B}_i^T) \sin\theta \end{pmatrix}, \\
\Gamma_{ij}^3 &= \begin{pmatrix} -rX & \tilde{A}_i X - \tilde{B}_i M_j \\ X \tilde{A}_i^T - M_j^T \tilde{B}_i^T & -rX \end{pmatrix}
\end{aligned} \tag{8}$$

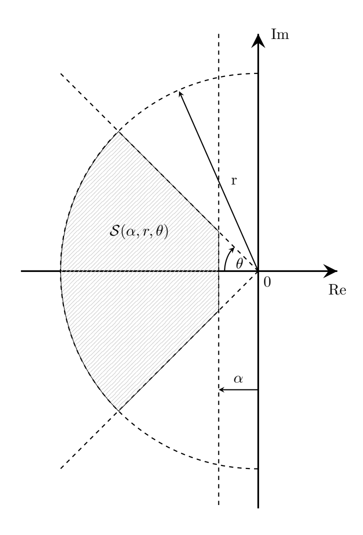


Figure 1: Parameterized pole region $S(\alpha, r, \theta)$

with the relaxed conditions

$$\begin{aligned}
&\Gamma^k_{ij}(X, M_j) + \Gamma^k_{ji}(X, M_i) \prec 0\,, \\
&\Gamma^k_{ii}(X, M_i) \prec 0\,, \quad k = 1,2,3 \quad \text{for all} \quad i = 1,2,\ldots,N_r\,, \quad j = i+1, i+2, \ldots, N_r \\
&\qquad\qquad s.t. \quad h_i(z)h_j(z) \neq 0,\ \exists z
\end{aligned}$$

So that, the feedback gains of the control law (3) can be obtained

$$\tilde{K}_j = M_j X^{-1}\,, \quad j = 1,2,\ldots,N_r\,. \tag{9}$$

An application of this concept in the stability and control of renewable-energy power plants has been demonstrated, i.e., for wind turbines [8] and PV power plants [9].

3 TS controller design by model matching using reference model specification

In comparison to the specification of a pole region as shown in Figure 1, for the second method the closed dynamics is specified by a reference model

$$\dot{x}^r = A^r x^r + E^r w\,, \qquad y = C^r x^r + F^r w\,. \tag{10}$$

The used process model distinguishes between the controller input u and an external reference w:

$$\dot{x} = \sum_{i=1}^{N_r} h_i(z)\big(A_i x + B_i u + E_i w\big), \quad y = \sum_{i=1}^{N_r} h_i(z)\big(C_i x + F_i w\big). \tag{11}$$

The related the control law is specified as

$$u = -\sum_{i=1}^{N_r} h_i(z) \underbrace{\big(K_{x,i}\,,\, K_{x^r,i}\,,\, -K_{I,i}\big)}_{\bar{K}_i} \underbrace{\begin{pmatrix} x \\ x^r \\ x_I \end{pmatrix}}_{\bar{x}} \tag{12}$$

with the same auxiliary state vector x_I proposed in (2). An augmented design model is specified using the extended state vector x (12) and matching error $\varepsilon = \dot{x}_I = y^r - y$:

$$\begin{aligned} \dot{\bar{x}} &= \sum_{i=1}^{N_r} h_i(z) \underbrace{\begin{pmatrix} A_i & 0 & 0 \\ 0 & A^r & 0 \\ -C & C^r & 0 \end{pmatrix}}_{\bar{A}_i} \bar{x} + \sum_{i=1}^{N_r} h_i(z) \underbrace{\begin{pmatrix} B_i \\ 0 \\ 0 \end{pmatrix}}_{\bar{B}_i} u + \sum_{i=1}^{N_r} h_i(z) \underbrace{\begin{pmatrix} E_i \\ E_i^r \\ F^r - F_i \end{pmatrix}}_{\bar{E}_i} w\,, \\ \varepsilon &= \underbrace{\begin{pmatrix} -C_i & C^r & 0 \end{pmatrix}}_{\bar{C}_i} \bar{x} + \underbrace{\begin{pmatrix} F^r - F_i \end{pmatrix}}_{\bar{F}_i} w \end{aligned} \tag{13}$$

which results directly from the combination of (2), (11), and (12). By specifying the matching error ε, an H_∞ problem can be formulated to calculate the controller gain $\bar{K}$. In this regard, the design objective is to minimize the input-to-output gain denoted as γ with respect to the input w and output ε:

$$\begin{aligned} &\text{minimize} && \gamma \\ &\text{subject to} && \sup_{\|w\|_2 \neq 0} \frac{\|\varepsilon\|_2}{\|w\|_2} \leq \gamma\,, \end{aligned} \tag{14}$$

where $\|\cdot\|_2$ denotes the L_2-norm. Controller coefficients $\bar{K}_i$, $i = 1,\ldots,N_r$ from (12) are calculated by solving the associated LMI problem

$$\Gamma_{ij} = \begin{pmatrix} \bar{A}_i X + X\bar{A}_i^T - \bar{B}_i \bar{M}_j - \bar{M}_j^T \bar{B}_i^T & \bar{E}_i & X\bar{C}_i^T \\ \bar{E}_i^T & -\gamma^2 I & \bar{F}_i^T \\ \bar{C}_i X & \bar{F}_i & -I \end{pmatrix}, \quad X \succ 0 \tag{15}$$

where $X = X^T$ with the relaxed condition

$$\begin{aligned}&\Gamma_{ij}(X,M_j)+\Gamma_{ji}(X,M_i) \prec 0\,,\\ &\Gamma_{ii}(X,M_i) \prec 0\,,\text{for all} \quad i=1,2,\dots,N_r\,, \quad j=i+1,i+2,\dots,N_r\,.\\ &\qquad\qquad s.t. \quad h_i(z)h_j(z) \neq 0,\ \exists z\end{aligned}$$

4 Illustrating example

To illustrate the model matching method, we now examine the following multivariable nonlinear system to be controlled

$$\dot{x} = \begin{pmatrix} -\frac{1}{2}\cos(\frac{\pi}{20}x_1) & \frac{1}{4} \\ \frac{1}{3} & -\frac{4}{5} \end{pmatrix} x + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} u + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} w\,, \quad y = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} x\,. \tag{16}$$

with $x = (x_1\,, x_2\,)^T$, $w = (\Delta f\,, \Delta v\,)^T$, and $y = (\Delta p\,, \Delta q\,)^T$. The plant model is intended to represent a generating unit in a power system with the measured grid frequency Δf and voltage change Δv at the point of common coupling (PCC). The plant output consists of the change of active power Δp and reactive power Δq injected into the grid. A controller (12) is to be found, which should match the plant model (16) to the reference model

$$\dot{x^r} = \begin{pmatrix} -\frac{1}{2} & 0 \\ 0 & -\frac{4}{5} \end{pmatrix} x^r + \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} w\,, \quad y^r = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} x^r\,. \tag{17}$$

The reference model (17) is thus designed to reproduce the desired dynamic characteristics of a generating unit for grid control purposes. To be able to use the LMI formulation (15) for the controller design, the plant model must first be converted into a Takagi-Sugeno form. For this we apply the sector nonlinearitiy approach [7], where a bounded function $f(x) \in [\min(f), \max(f)]$ is described by

$$f(x) = \underbrace{\frac{\max(f) - f(x)}{\max(f) - \min(f)}}_{h_1(x)} \min(f) + \underbrace{\frac{f(x) - \min(f)}{\max(f) - \min(f)}}_{h_2(x)} \max(f) \tag{18}$$

Considering the practical limitation of the state space $x_1 \in [-4,4]$, the min-max values for the example (16) with $f(x) = \cos(\frac{\pi}{20}x_1)$ are calculated by

$$\min(f) = \cos(\frac{\pi}{20}\max(x_1))\,, \quad \max(f) = \cos(\frac{\pi}{20}0) = 1\,. \tag{19}$$

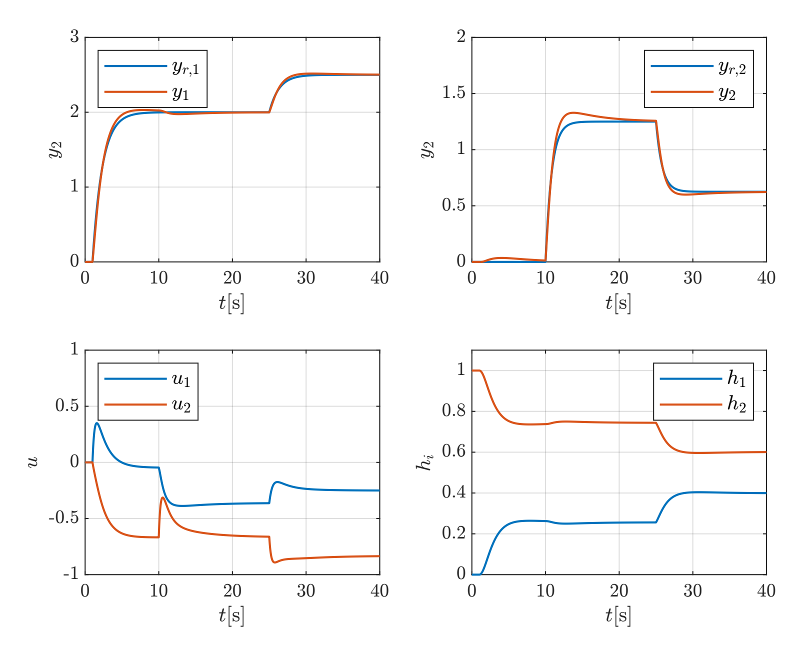


Figure 2: Simulation results using the model matching method

This results in the matrices of the submodels

$$A_1 = \begin{pmatrix} -\frac{1}{2}\max(f) & \frac{1}{4} \\ \frac{1}{3} & -\frac{4}{5} \end{pmatrix}, \quad A_2 = \begin{pmatrix} -\frac{1}{2}\min(f) & \frac{1}{4} \\ \frac{1}{3} & -\frac{4}{5} \end{pmatrix} \tag{20}$$

and by adding the membership functions function (18) we obtain

$$\dot{x} = \sum_{i=1}^{2} h_i(z) A_i x + B u, \qquad y = C x \tag{21}$$

with the common $B = B_1 = B_2$ and $C = C_1 = C_2$. After this preliminary work is done, the augmented model according to (13) can now be created, and the LMI problem (15) formulated. The simulation result for a unit step of Δf at $t = 1$s and of Δv at $t = 10$s is given in Figure 2.

5 Conclusion

Two methods for specifying the desired closed-loop dynamic for multivariable Takagi-Sugeno systems were presented. The necessary design steps were introduced and compared. A simplified example from the area of power plant control was used for the model matching procedure. In ongoing work, we are investigating how to integrate time-variable reference models into the design. The novel concept would address new design possibilities for fast coordination of power plants in interconnection with dynamic participation factors proposed in [4].

6 Acknowlegements

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 883985 (POSYTYF – POwering SYstem flexibiliTY in the Future through RES). https://posytyf-h2020.eu/

References

[1] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear Matrix Inequalities in System and Control Theory*, Philadelphia, PA, USA: SIAM, 1994.

[2] M. Chilali and P. Gahinet. *H_∞ design with pole placement constraints: An LMI approach*. IEEE Transactions on Automatic Control, vol. 41, no. 3, pp. 358-367, 1996.

[3] G. Feng, *A Survey on Analysis and Design of Model-Based Fuzzy Control Systems*, IEEE Transaction on Fuzzy Systems, vol. 14, no. 5, October 2006.

[4] V. Häberle, M. W. Fisher , E. Prieto-Araujo, F. Dörfler. *Control Design of Dynamic Virtual Power Plants: An Adaptive Divide-and-Conquer Approach*, IEEE Transaction on Power Systems, vol. 37, no. 5, September 2022.

[5] B. Marinescu, O. Gomis-Bellmunt, F. Dörfler, H. Schulte and L. Sigrist. *Dynamic Virtual Power Plant: A New Concept for Grid Integration of Renewable Energy Sources*, IEEE Access, September 2022, doi: 10.1109/ACCESS.2022.3205731.

[6] A.T. Nguyen, T. Taniguchi, L. Eciolaza, M. Sugeno. *Fuzzy Control Systems: Past, Present and Future*, IEEE Computational Intelligence Magazine, vol: 14, no. 1, February 2019.

[7] H. Ohtake, K. Tanaka, and H. O. Wang. *Fuzzy Modeling via Sector Nonlinearity Concept*. In Joint 9th IFSA World Congress and 20th NAFIPS International Conference, pp. 127-132, Vancouver, Canada, 2001.

[8] F. Pöschke, V. Petrovic, F. Berger, L. Neuhaus, M, Hölling, M. Kühn, and H. Schulte, *Model-based wind turbine control design with power tracking capability: a wind-tunnel validation*, Control Engineering Practice, vol. 120, March 2022

[9] S. Kusche, H. Schulte, *Demanded Power Point Tracking of PV Power Plants without Battery Energy Storage*, Proc. 31st Workshop on Computational Intelligence, pp. 169-187, 2022.

[10] K. Tanaka and H.O. Wang. *Fuzzy Control Systems Design and Analysis: A Linear Matrix Inequality Approach*. John Wiley & Sons, Inc, 2001.

Dieser Tagungsband enthält die Beiträge des 32. Workshops „Computational Intelligence" des Fachausschusses 5.14 der VDI/VDE-Gesellschaft für Mess- und Automatisierungstechnik (GMA) und der Fachgruppe „Fuzzy-Systeme und Soft-Computing" der Gesellschaft für Informatik (GI), der vom 1.12. – 2.12.2022 in Berlin stattfindet.

Der GMA-Fachausschuss 5.14 „Computational Intelligence" entstand 2005 aus den bisherigen Fachausschüssen „Neuronale Netze und Evolutionäre Algorithmen" (FA 5.21) sowie „Fuzzy Control" (FA 5.22). Der Workshop steht in der Tradition der bisherigen Fuzzy-Workshops, hat aber seinen Fokus in den letzten Jahren schrittweise erweitert.

Die Schwerpunkte sind Methoden, Anwendungen und Tools für
- Fuzzy-Systeme,
- Künstliche Neuronale Netze,
- Evolutionäre Algorithmen und
- Data-Mining-Verfahren

sowie der Methodenvergleich anhand von industriellen und Benchmark-Problemen.

Die Ergebnisse werden von Teilnehmern aus Hochschulen, Forschungseinrichtungen und der Industrie in einer offenen Atmosphäre intensiv diskutiert. Dabei ist es gute Tradition, auch neue Ansätze und Ideen bereits in einem frühen Entwicklungsstadium vorzustellen, in dem sie noch nicht vollständig ausgereift sind.


ISBN 978-3-7315-1239-4

Gedruckt auf FSC-zertifiziertem Papier